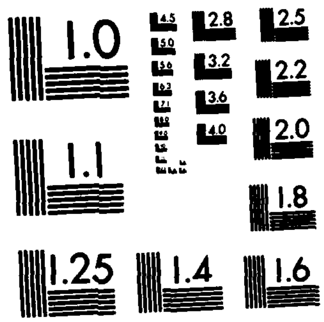
AD-A123 146 INFLUENCES OF HARDWARE IMPLEMENTATION ON A HIGH SPEED DIGITAL ADAPTIVE FI..(U) CALIFORNIA UNIV DAVIS SIGNAL AND IMAGE PROCESSING LAB K D WEINMANN APR 82 SIPL-82-4 1/2

UNCLASSIFIED AFOSR-TR-82-1085 AFOSR-80-0189 F/G 9/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFOSR-TR- 82-1085

UCD

University of California, Davis

AD A123146

DTIC FILE COPY

THE UNIVERSITY OF CALIFORNIA LET THERE BE LIGHT 1868

SIGNAL AND IMAGE PROCESSING LAB

83 01 07 025

3

INFLUENCES OF HARDWARE IMPLEMENTATION ON A
HIGH SPEED DIGITAL ADAPTIVE FILTER
USING THE RESIDUE NUMBER SYSTEM

by

Kimberly D. Weinmann

Report No. SIPL-82-4 April 1982

This work is supported by the United States Air Force
Office of Scientific Research under Grant #80-0189

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 82-1085	2. GOVT ACCESSION NO. AD-A123 146	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) INFLUENCES OF HARDWARE IMPLEMENTATION ON A HIGH SPEED DIGITAL ADAPTIVE FILTER USING THE RESIDUE NUMBER SYSTEM		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL
		6. PERFORMING ORG. REPORT NUMBER SIPL-82-4
7. AUTHOR(s) Kimberly D. Weinmann		8. CONTRACT OR GRANT NUMBER(s) AFOSR-80-0189
9. PERFORMING ORGANIZATION NAME AND ADDRESS Signal & Image Processing Laboratory, Department of Electrical & Computer Engineering, University of California, Davis CA 95616		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A1
11. CONTROLLING OFFICE NAME AND ADDRESS Mathematical & Information Sciences Directorate Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE Apr 1982
		13. NUMBER OF PAGES 94
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A study of the performance and certain characteristics of an eight-weight adaptive FIR digital filter implementing the LMS algorithm using Residue Number System (RNS) arithmetic hardware has been done in order to draw conclusions as to the optimum hardware configuration. Affects of the hardware and unknown system on the rate of convergence and adaptive algorithm step size were found by running filter simulations for different values of the system quantities. It was found that characteristics of the unknown system (or plant) do not affect the choice of filter hardware. The optimum step size for use in the hardware

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONTINUED: was found. Suggestions were made for further hardware improvement using a sign-magnitude system as opposed to 2's complement.

ABSTRACT

A study of the performance and certain characteristics of an eight-weight adaptive FIR digital filter implementing the LMS algorithm using Residue Number System (RNS) arithmetic hardware has been done in order to draw conclusions as to the optimum hardware configuration. Affects of the hardware and unknown system on the rate of convergence and adaptive algorithm step size were found by running filter simulations for different values of the system quantities. It was found that characteristics of the unknown system (or plant) do not affect the choice of filter hardware. The optimum step size for use in the hardware was found. Suggestions were made for further hardware improvement using a sign-magnitude system as opposed to 2's complement.

ACKNOWLEDGMENT

This culmination of the author's education would not have been possible without the support of her parents. A special thanks goes to them for their support; mental, monetary, and otherwise. Considering his busy schedule, thanks also go to Dr. Michael A. Soderstrand for his continuous help.

CONTENTS

CHAPTER I. INTRODUCTION

1.0 Introduction

Digital signal processing is a dynamic, rapidly growing field, but its fundamentals are well established [1,2]. The techniques and applications of digital signal processing are expanding at a tremendous rate. With the advent of large scale integration and the resulting reduction in cost and size of digital components together with increasing speed, the number of applications of digital signal processing techniques is growing. Special purpose digital filters can now be implemented in the megahertz range, and simple digital filters have been integrated on circuit chips. Digital processors also form an integral part of many modern radar and sonar systems. When digital filters are coupled with the advantages of adaptive systems the results can be very exciting [3,4]. Adaptive filters have distinct advantages over fixed parameter and operator adjustable systems for many applications. Fortunately, most digital filters can be made adaptive through the use of an adaptive updating algorithm.

At UCD over the past several years graduate students have been studying digital adaptive filtering under a contract from the United States Air Force [5]. One project funded under this contract was to simulate and build a digital adaptive filter which would run at a very high sampling rate. This filter is discussed in a paper by M.A. Soderstrand and J.K. Kelley [6] in which a hardware design is suggested and a report is made on the filter simulation.

The purpose of this thesis is to further develop this hardware and to study the performance and certain characteristics of this computer simulation as they apply to the hardware design of the filter. (See Appendices A and B for computer program and simulated system diagram.) Specifically, we will:

1. Firm up the hardware design for the adaptive part of the system.
2. Carry out a detailed simulation of this adaptive hardware.
3. Draw conclusions as to the optimum adaptive hardware configuations.

During the process of this study our focus will remain on the hardware implementation (to be carried out at some later date). Thus our choice for filter structure and adaptive algorithm are very much dependent upon the fact that this filter will be built and not merely simulated.

2.0 FIR Filter Hardware

2.1 Pipelined FIR Filter

The filter structure chosen for our adaptive filter is the 8-weight pipelined Tapped Delay Line filter (PTDL) shown in Figure 1a [7] which evolved from the classical Tapped Delay Line filter (TDL) of Figure 1b.

The PTDL of Figure 1a has a sampling rate 7 times faster than that of the TDL of Figure 1b due to the parallel processing of all the partial sums with each other. From straightforward analysis the difference equation of the PTDL filter is

$$y(i) = \sum_{j=o}^{7} a_j x(i-2-j).$$

This can be compared to the difference equation of the TDL which is

$$y(i) = \sum_{j=o}^{7} a_j x(i-j).$$

It is clear that the effect of pipelining is to delay the output by two time samples. The PTDL filter could be extended from 8 to any desired number of filter weights.

2.2 RNS Implementation

The Residue Number System (RNS) becomes extremely useful in the hardware implementation of the digital filter [8-16]. We have chosen the moduli 11, 13, 15, and 16 due to the range of numbers required. The PTDL is implemented in modular arithmetic for each of the four moduli in parallel. This parallel structure requires no arithmetic carries as would be required in a binary system. The sampling rate of an RNS filter can thus be much faster than that of a binary system [17].

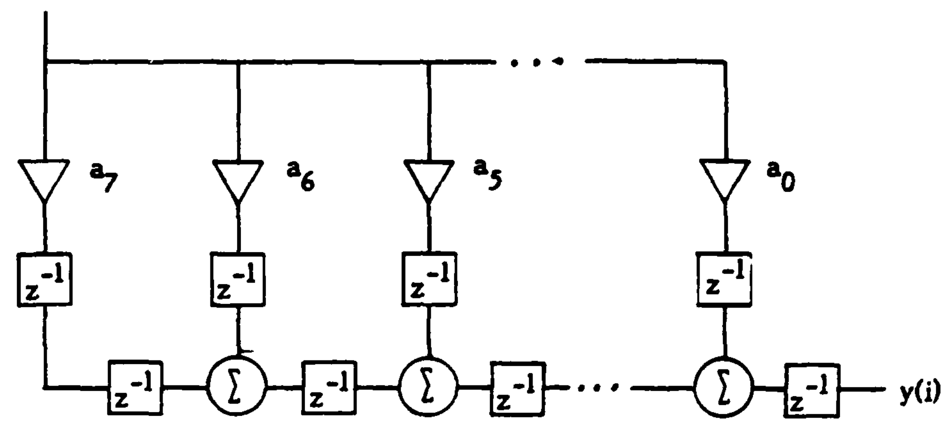


Figure 1a
Pipelined Tapped Delay Line Filter (PTDL)

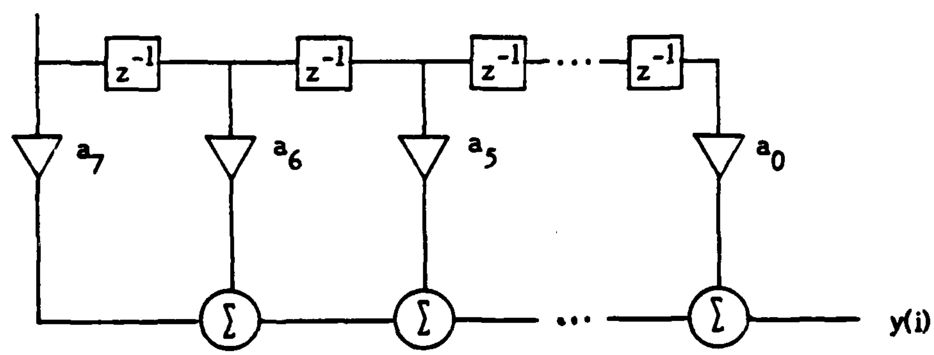


Figure 1b
Tapped Delay Line Filter (TDL)

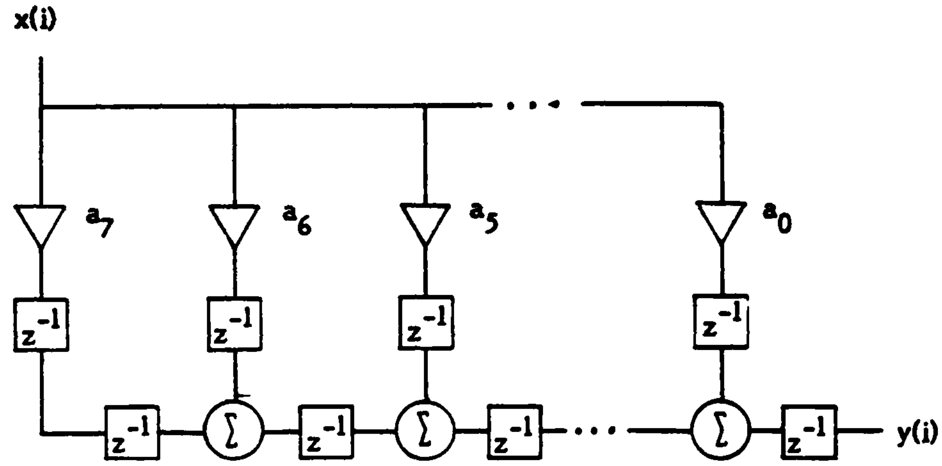


Figure 1a
Pipelined Tapped Delay Line Filter (PTDL)

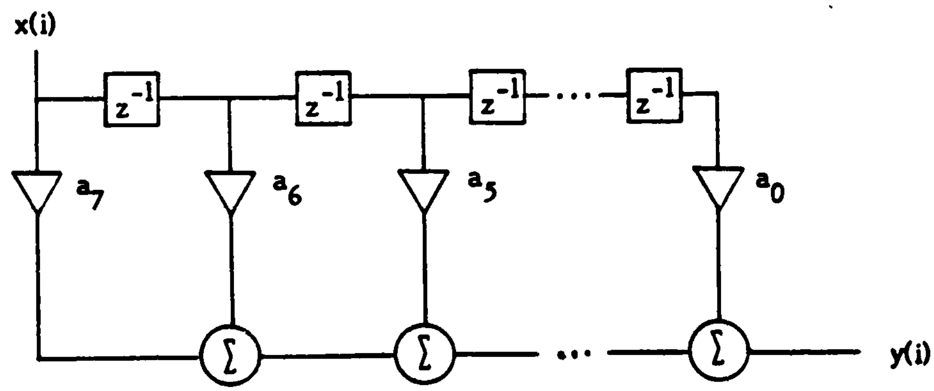


Figure 1b
Tapped Delay Line Filter (TDL)

In practice one must generally convert binary to residue for processing and after processing convert back to binary. The conversion from binary to residue is quite simple, usually done by straight table look-up [18]. The conversion back, although somewhat more complex, is relatively simple. Several techniques exist for this RNS to binary convergence including mixed radix conversion [18-20] and conversion based on the Chinese reminder theorem [19,21].

Figure 2 shows the basic hardware for one modulus of the digital filter. The hardware for each modulus is identical except for the arithmetic tables stored in the ROMs. Each weight is implemnted by a 256x4 ROM with 4 of the 8 address bits selected by the modulus m_k weight a_j. Each adder is implemented by a 256x4 ROM with the 8-bit address selected by the two 4-bit modulus m_k numbers to be added. Each moduli of the FIR filter requires 2n-1 ROMs and 2n delays for n weights. For our 8-weight filter each modulus has 15 ROMs and 16 delays, resulting in a total of 60 ROMs and 64 delays.

2.3 Adaptive Filter

This PTDL filter can now be used in an adaptive system. An adaptive filter structure must be chosen with which to test the filter, and an adaptive algorithm must be chosen to update the filter weights.

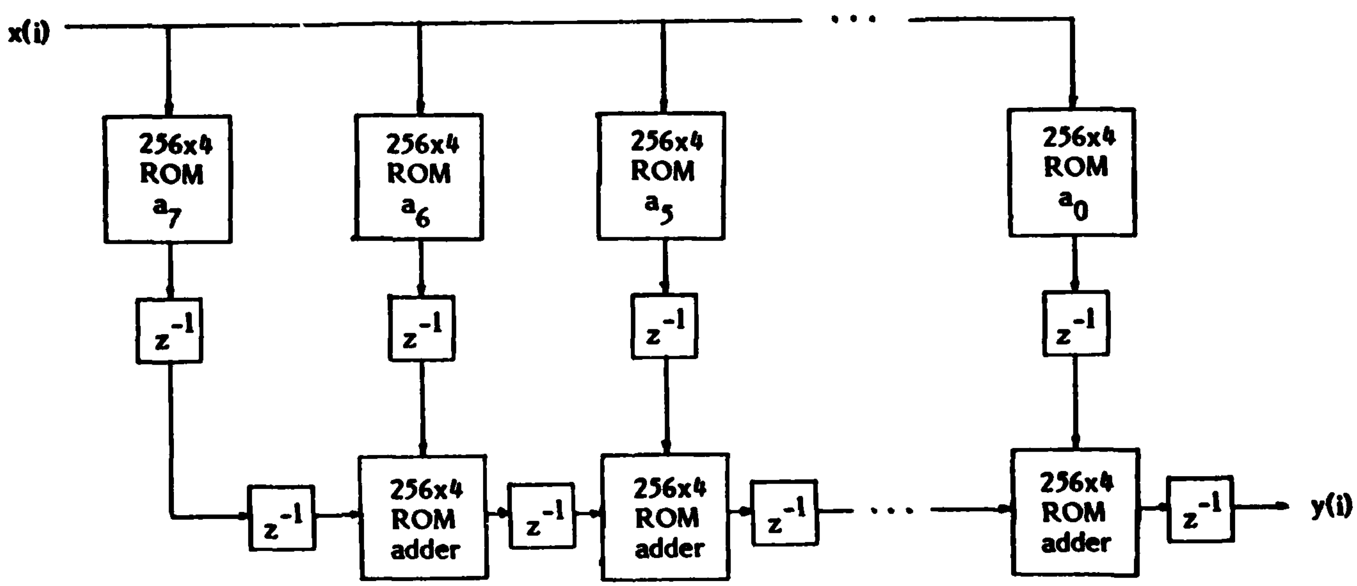


Figure 2
Hardware For One Modulus of Digital Filter

3.0 Adaptive Filter Design

3.1 Adaptive Filter Structure

Adaptive filtering is a very useful tool in filter design. Adaptive filters can be used in many different structures to perform many different tasks. We would like to choose one of these structures in which to test our filter. Most uses for adaptive filters can be grouped into four categories:

(1) System Identification (Figure 3)

(2) Noise Cancellation (Figure 4)

(3) Channel Enhancement (Figure 5)

(4) Model Reference (Figure 6)

The adaptive filter in each configuration is a separable addition to the original system. From the viewpoint of the adaptive filter the system which contains it is a black box. The filter receives signals from the system and outputs signals to it. An adaptive filter will work independently of the type of system it is in. Since the purpose of this paper is to study adaptive filtering and not uses for adaptive filters, we have chosen the System Identification configuration with which to test our filter. System ID is the simplest of the four configurations and thus simplifies our study.

3.2 Adaptive Algorithm

Adaptive systems adapt by means of minimizing (or optimizing) some system parameter. This parameter is measured by an Index of Performance (IP) function frequently expressed in the form

$$J(c) = \int_0^{\infty} f(e,c,t)\,dt$$

where e is the error representing the deviation of the system parameter from the desired value, c is a set of independently adjustable variables, and t is time.

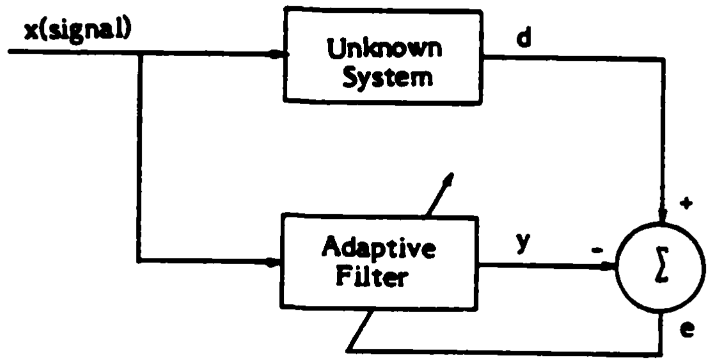


Figure 3
System Identification

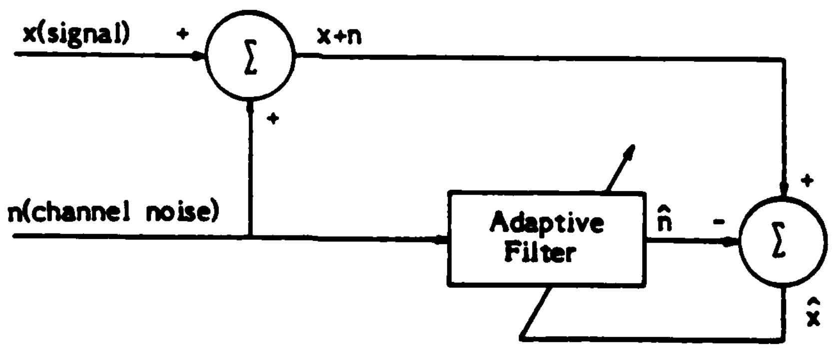


Figure 4
Noise Cancellation

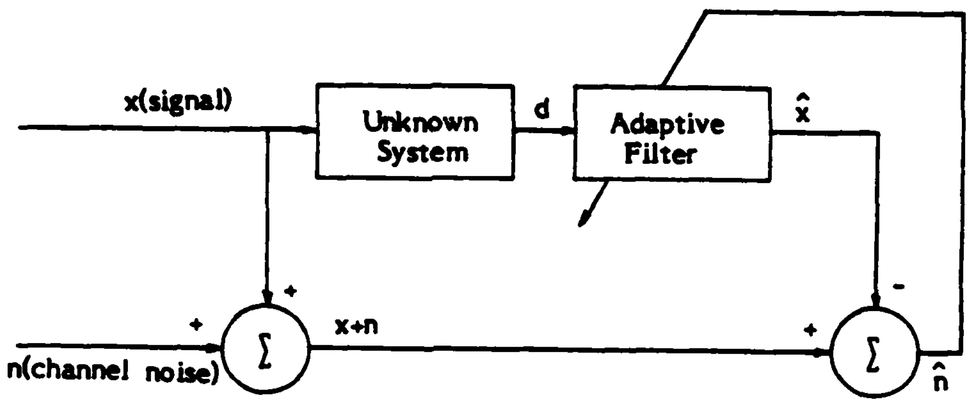


Figure 5
Channel Enhancement

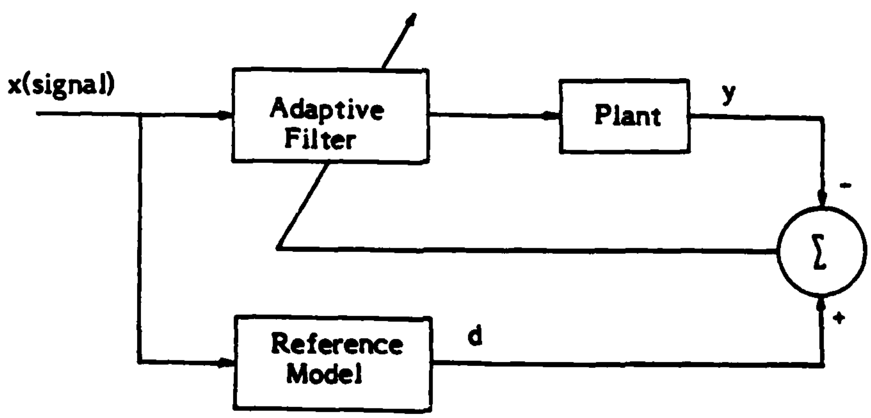


Figure 6
Model Reference

Each of the four configurations in which adaptive filters are used has an 'error' e which is used by an IP function. For System Identification e is the difference between the unknown and the adaptive filter outputs. The error e for Noise Cancellation systems is the approximate input signal x which approaches x as adaptation cancels the noise. Channel Enhancement error e is the difference between the original and channel-distorted signals n and should approach zero. In Model Reference systems e is the deviation of the actual system output from the ideal output. For each system, c is the set of adaptive filter weights.

There are many possible IP functions which we could choose to minimize in our adaptive filter. Only the two most common, Least Mean Squares and Least Squares, were considered for this study.

3.2.1 Least Mean Squares

The first of these two Index of Performance functions is the Least Mean Squares (LMS) in which $J(w)=E[e_j^2]$ is minimized [22]. In this case c (the independent variables) are the unknown filter weights w. The System Identification configuration for our system looks like that in Figure 7. The error at time j is

$$e_j = d_j - y_j = d_j - x_{j-2}^T w.$$

The square of the error is

$$e_j^2 = d_j^2 - 2d_j\, x_{j-2}^T w + w^T x_{j-2}\, x_{j-2}^T w.$$

The mean square error ξ (which equals $J(w)$) is

$$\xi \triangleq E[e_j^2] = E[d_j^2] - 2\,E[d_j\, x_{j-2}^T]\, w + w^T E[x_{j-2}\, x_{j-2}^T]\, w$$

$$= E[d_j^2] - 2\, P^T w + w^T R\, w.$$

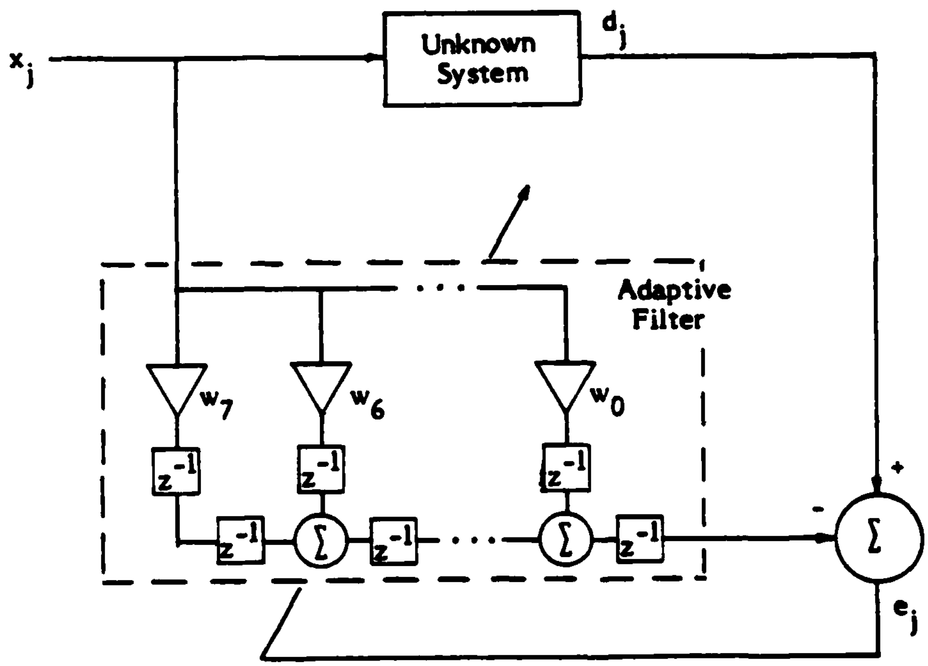


Figure 7
System Identification with PTDL

where P is defined as the cross correlation vector between the input signal and the desired response, $E[d_j^T x_{j-2}]$, and R is the input correlation matrix, $E[x_{j-2}\, x_{j-2}^T]$. ξ is a quadratic function of the adaptive filter weights, and has an n-dimensional bowl shape as shown in Figure 8.

ξ can be minimized by means of a gradient search using the following steepest descent recursive algorithm:

$$w_{j+1} = w_j + \mu\,(-\nabla_j)$$

where
$$\nabla_j = \frac{\partial \xi}{\partial w}\Big|_{w\,=\,w_j} = -2P + 2R\, w_j$$

and μ is the step size.

This recursive equation is known as the Least Mean Squares (LMS) algorithm. In practice this form of the algorithm is not useful because P and R are not known. An estimate of the mean square error is $\xi \approx [e_j^2]$ which gives us as approximate gradient

$$\hat{\nabla}_j = -2e_j\, x_j.$$

The approximate LMS algorithm is then

$$w_{j+1} = w_j + 2\mu e_j\, x_j$$

which is very easy to apply in practice.

There are variations of the LMS algorithm which can converge faster than the LMS. The most popular of these is the Normalized LMS (NLMS) with updating formula

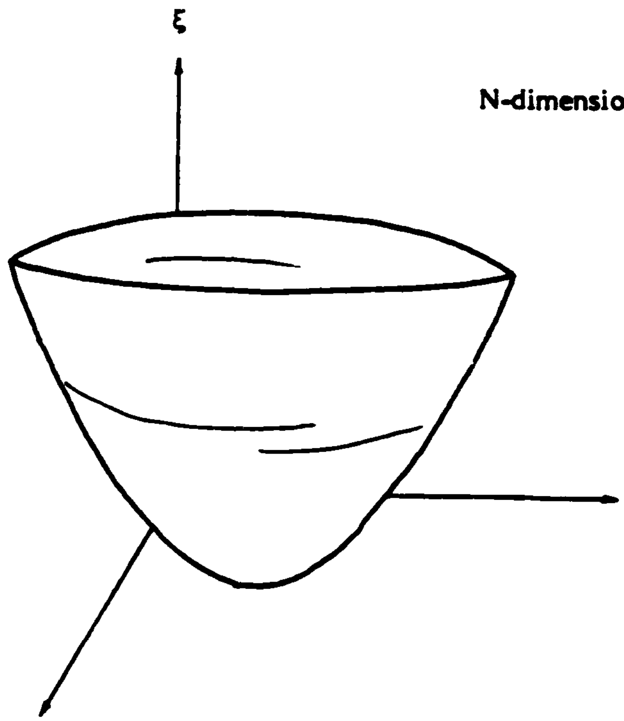


Figure 8
Mean Square Error Function

$$w_{j+1} = w_j + \frac{\alpha}{||x_j||^2} e_j\, x_j.$$

This algorithm will converge faster than the LMS, but requires more hardware to be implemented.

3.2.2 Least Squares

The second important Index of Performance function is the Least Squares where $\mathcal{J}(w) = \sum_j e_j^2$ is minimized. In System Identification the problem is to determine the unknown weights. This is done in a method similar to that of the LMS update equation. This recursive formula is

$$w_{j+1} = w_j + \gamma_{j+1} P_j\, x_{j+1} [y_{j+1} - x_{j+1}^T w_j]$$

where
$$P_{j+1} = P_j - \gamma_{j+1} P_j\, x_{j+1}\, x_{j+1}^T P_j$$

and
$$\gamma_{j+1} = 1/[1 + x_{j+1}^T P_j\, x_{j+1}]$$

Therefore, by starting with an initial estimate w_o and P_o, w can be sequentially updated while new observations are continuously obtained.

3.2.3 Choice of Algorithm

The main criterion used to choose an adaptive algorithm for our system was complexity of hardware required. This is due to limited board space and cost factors of the filter. A key factor in the hardware selection is the fact that in order to update n weights, it takes n^2 operations [17]. Thus for a practical number of weights, each update operation must be very simple.

Contrary to what might be expected, number of iterations for convergence was not a major criterion used to choose an algorithm. Input signals to the filter are assumed to be in the audio range. The basic sampling rate is 10MHz, hence 10,000 iterations can be made in one millisecond. If this were not the case, number of iterations for convergence might alter the choice of algorithm.

Given the criterion for simple hardware the obvious choice of adaptive algorithm for our filter is the Least Mean Squares. It requires relatively simple hardware and is fast enough for our purposes.

3.2.4 Hardware Implementation of LMS Algorithm

The hardware chosen and simulated by J.K. Kelley for the LMS adaptive algorithm is shown in Figure 9. Eight bits are available with which to represent the input x and the error e. The 8 bits must somehow be divided between x and e. Here we can see that a hardware implementation can considerably decrease the accuracy of an adaptive algorithm. The number of bits allowed for x is called NXBIT. One bit of 8 is used for the sign of (e)(x) leaving 7-NXBIT bits to represent e.

The purpose of this thesis is to study this division of the bits between e and x. We will find which division gives the fastest convergence and how the optimum step size is affected by the division. We will also show how the rate of convergence and optimum step size are affected when the adaptive and unknown filters have different numbers of weights.

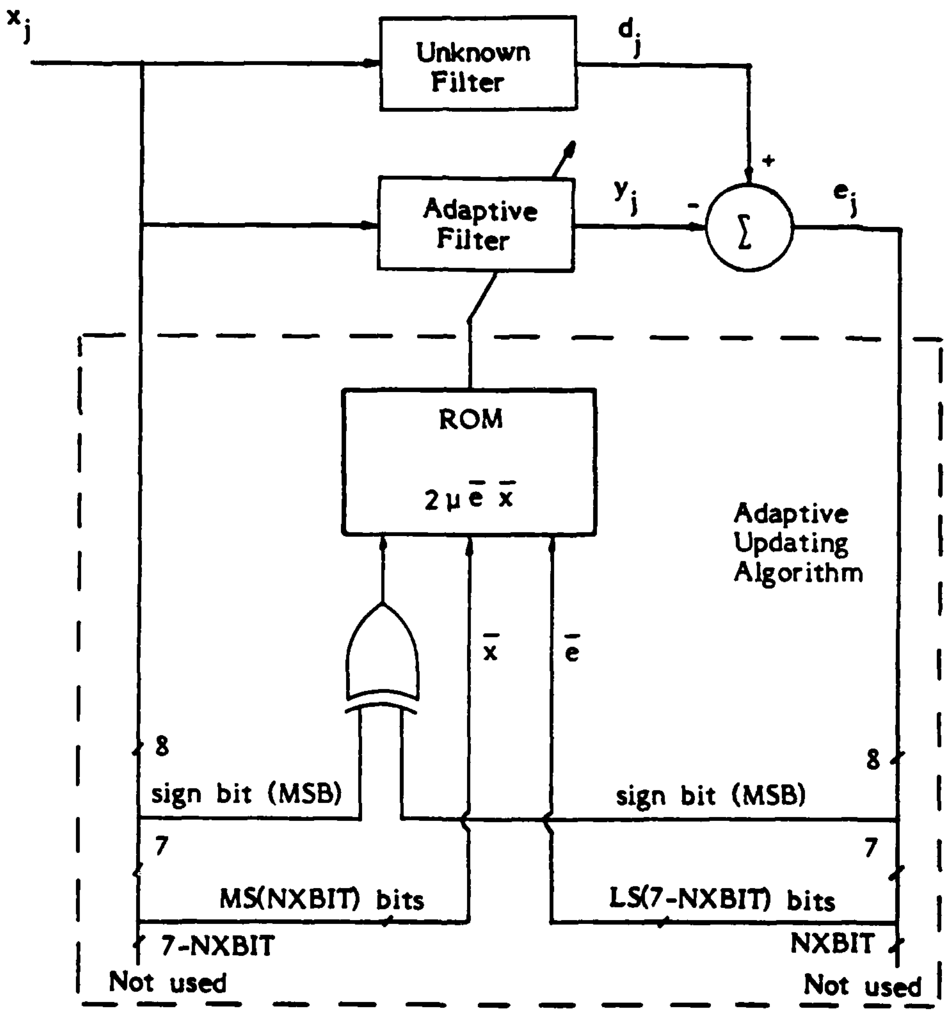


Figure 9
Adaptive Filter with Updating Algorithm

CHAPTER II. PROCEDURE

1.0 Introduction

As discussed in Chapter I, the purpose of this thesis is to study how the rate of convergence and optimum step size μ^* are affected by:

i) the division of bits between e and x

ii) the number of weights in the unknown filter

The results of this study will define a μ_o and NXBIT for optimum convergence to be used in the adaptive filter hardware shown in Figure 9.

Figure 9 shows the update hardware for the digital adaptive filter. A more detailed picture of the hardware is shown in Figure 10a for the case when 3 bits of the input x and 4 bits of the error e are used to calculate the weight adjustment vector $2\mu_o ex$. The update quantity is calculated in the ROM and assumes that e and x are in 2's complement binary form. Thus the ROM needs the sign of (e)(x) in order to calculate the correct update quantity. This leaves 7 bits to divide between e and x.

A different, and probably better, hardware implementation is discussed in Chapter III which assumes e and x are in sign-magnitude binary form. For this case the sign of (e)(x) need not be fed into the ROM, but can multiply the result, leaving 8 bits to divide between e and x. This method is shown in Figure 10b.

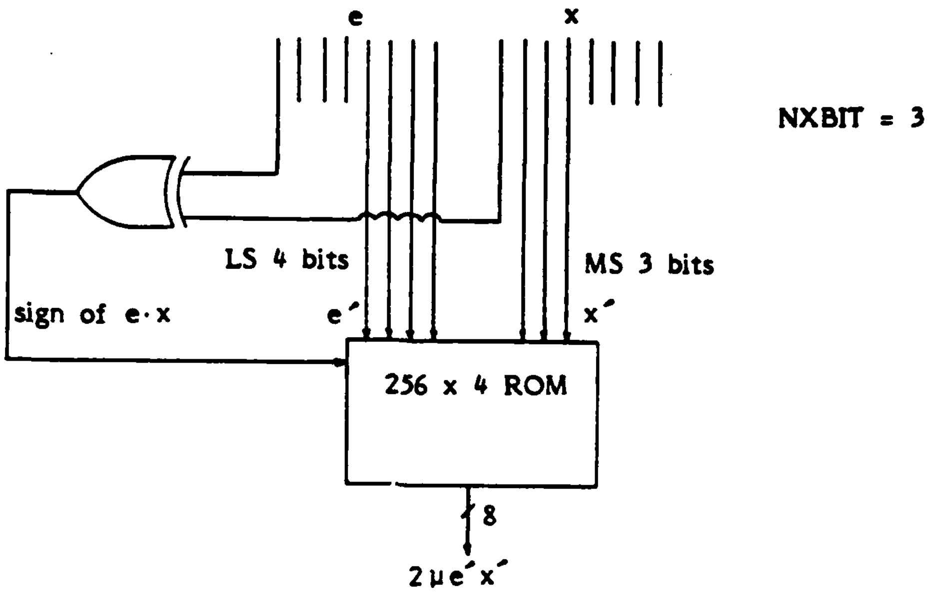


Figure 10a
Update Hardware Using 2's Complement Numbers

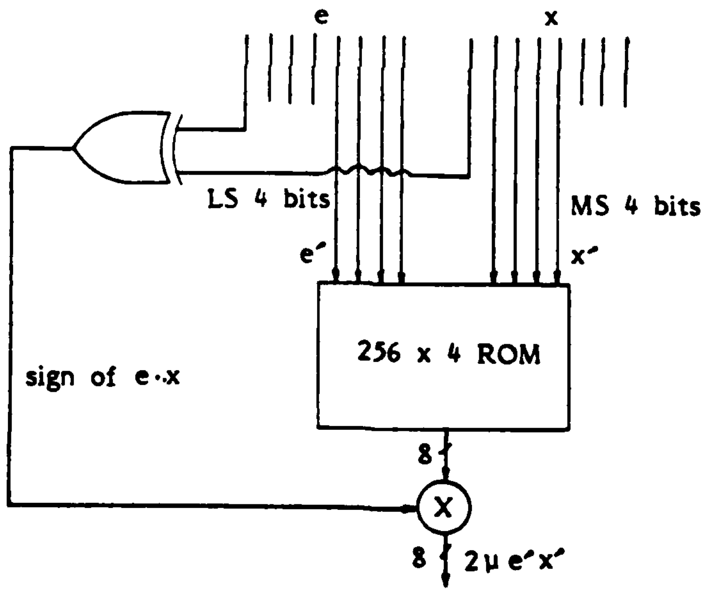


Figure 10b
Update Hardware Using Sign-Magnitude Numbers

2.0 Optimum Step Size

The Least Mean Squares adaptive algorithm updates the adaptive filter weights using the recursive formula $w_{j+1} = w_j + 2\mu e_j x_j$, where μ is the step size. In general, two aspects of the step size are of interest: the maximum allowable gain μ_{max} for stability, and the optimum μ^* for fastest convergence [23].

It has been shown that for the LMS algorithm to converge [22] μ must be bounded as

$$0 < \mu < \mu_{max}$$

where

$$\mu_{max} = \frac{2}{trR} = \frac{2}{E[||x_k||^2]} \, . \tag{1}$$

As seen in Chapter I, R is the input correlation matrix defined by

$$R \triangleq E[x_k x_k^T] \, .$$

Gitlin and Weinstein [24] showed that the μ which achieves maximum rate of convergence μ^* is

$$\mu^* \approx \tfrac{1}{2} \mu_{max} \, . \tag{2}$$

In practice R is not known and another more practical form of the equation μ^* must be found. In work recently done by Gardner [25] a more practical form is obtained:

$$\mu_{max} = \frac{2}{(N+2)\sigma^2} = 2\mu^* \qquad (3)$$

The input vector x_k is assumed to be Gaussian with independent and identically distributed elements. N is the order of the adaptive filter.

The adaptive filter we have used is 7th order and the standard deviation (σ) of the input signal used for simulation is .30. According to equation 3 we should ideally have

$$\mu_{max} = 2.46 = 2\mu^*.$$

The Residue Number System requires that the non-integer input signal be scaled by the factor SCALE, which for our system is 130. (See Appendix B for calculation of SCALE.) This acts to divide the step size by SCALE so that the ideal optimum step size will be

$$\mu^{*\prime} = \mu^*/\text{SCALE} = .0094$$

2.1 Determination of Optimum Step Size

In the last section we discussed the ideal optimum step size μ^* and gave equations to calculate it. In order to find the optimum step size μ_o to use in our hardware, a strategy must be designed with which to obtain the optimum μ from the data output of our computer simulation. Our simulation plots the ensemble averaged output error. An ensemble averaged curve is simply the average of a number of such individual curves and approximates the adaptive behavior in the mean.

The optimum step size is the value of μ which minimizes the mean square error:

$$\frac{\sum_{j=1}^{k} e_j^{\,2}}{K}$$

This is a commonly used technique which is consistent with the fact that we have chosen the Least Mean Squares adaptive algorithm which also minimizes the mean square error. K is the number of iterations chosen to average over and is larger than the number of iterations required for the error curve of each μ value to settle. Ideally K would be infinity, but fortunately we may adequately estimate μ^* with a relatively small K (approximately 1,000 for our case). This method of finding μ^* is very easy to implement in our simulation.

The ensemble averaged output error ideally has the form of an exponential:

$$e(t) = k_1 e^{-\alpha t}$$

In practice, however, the output error has an 'error floor' that is due to hardware approximations. This error floor is represented by k_2 in the non-ideal exponential form of output error:

$$e(t) = k_1 e^{-\alpha t} + k_2$$

The output error curves cannot drop below the error floor, therefore, the number of iterations to average K can be determined by observing when the error curves have settled to k_2.

2.2 Effects of Truncating x and Limiting e

As discussed in section 3.2.4 of Chapter I, 8 bits are available with which to represent the input, the error, and the sign of the product of input and error in the adaptive algorithm. The approximation for x x' is found by simply truncating x to NXBIT bits. The approximation for e e′, however, is found by saturating at e=2**(7-NXBIT)-1 if e is too large to be represented with 7-NXBIT bits. This is done in order to obtain sensitive adaptation near convergence. Plots of x and e as they are approximated to x′ and e′ are shown in Figures 11a-e. These approximations will affect the error floor and the rate of convergence of our simulations.

It can be shown that the rate of convergence is affected by both the saturation of e and the truncation of x. However, as convergence is approached the error becomes small and is thus no longer saturated.

Similarly, saturation of e does not affect the error floor because as the error floor (convergence) is reached e is not saturated. Truncation, however, has an effect on the error floor, but its effect may be masked by finite arithmetic errors which are due to integer arithmetic used in the filter.

These facts will be supported with data in Chapter III. In particular, we shall see that the error floor is primarily determined by the finite arithmetic and that the rate of convergence is mainly affected by the saturation of e and the truncation of x.

2.3 Filter Order Mismatching Error

Part of this study is to make conclusions on how rate of convergence and optimum step size are affected by the number of weights in the unknown filter. The adaptive filter used will have 8 weights (7th order), and for simulation any number of weights can be entered for the unknown filter. However, when the filter is built and used the unknown filter will be just that, unknown, and may have any number of weights. For this reason we will simulate the hardware using unknown filters of 7, 8 and 9 weights. This will enable us to draw conclusions about unknown filters of less than, equal to, and greater than 8 weights.

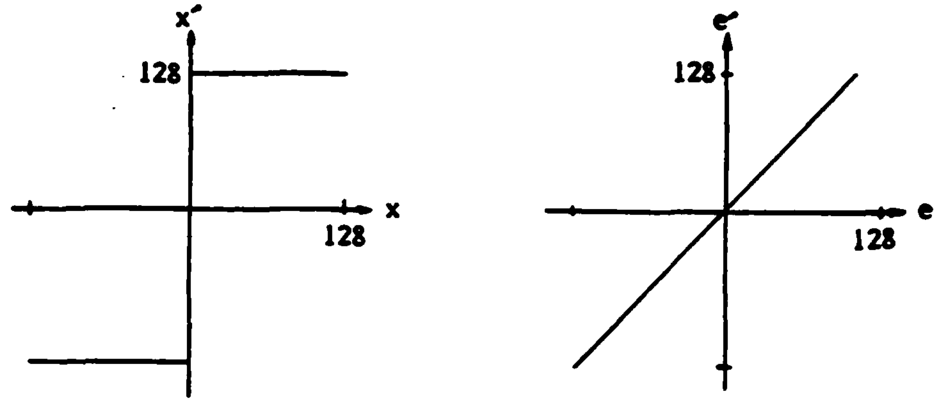


Figure 11a
Approximations of x and e - NXBIT = 0

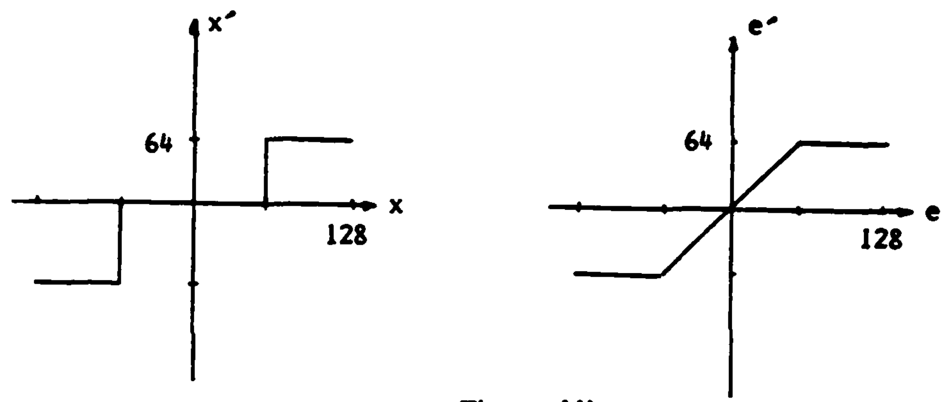


Figure 11b
Approximations of x and e - NXBIT = 1

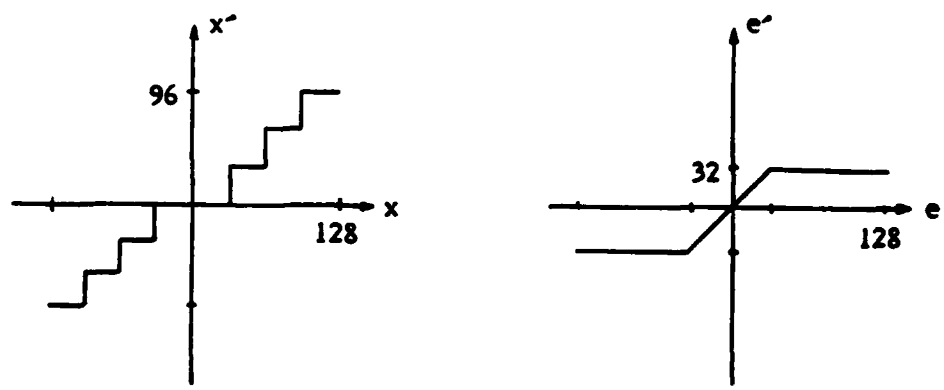


Figure 11c
Approximations of x and e = NXBIT = 2

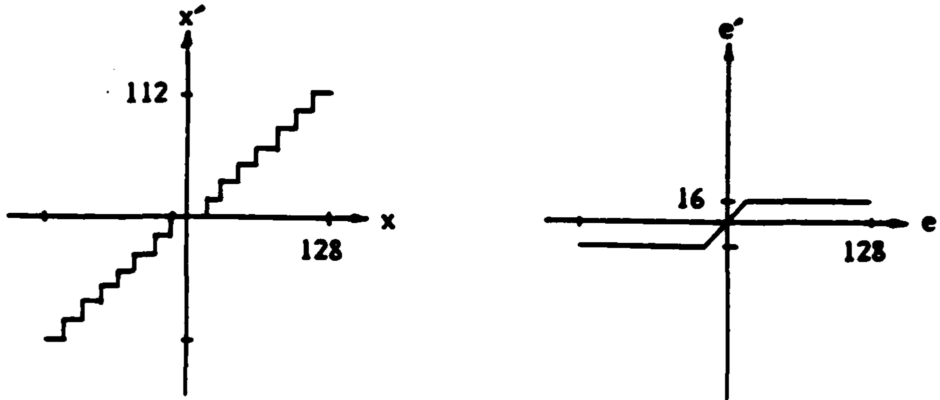


Figure 11d
Approximations of x and e - NXBIT = 3

(NXBIT = 4,5 not shown, but follow pattern)

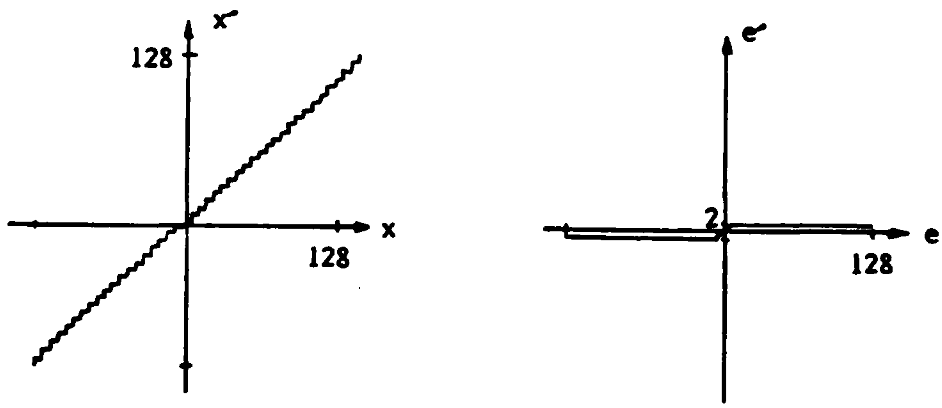


Figure 11e
Approximations of x and e = NXBIT = 6

For this simulation we have chosen the FIR lowpass filters with all zeros at $z = -1$ on the unit circle. This choice of unknown filters is essentially arbitrary, although our goal is to choose filters wtih similar properties. The transfer functions are:

7 weights: $z^6 + 6z^5 + 15z^4 + 20z^3 + 15z^2 + 6z + 1$

8 weights: $z^7 + 7z^6 + 21z^5 + 35z^4 + 35z^3 + 21z^2 + 7z + 1$

9 weights: $z^8 + 8z^7 + 28z^6 + 56z^5 + 70z^4 + 56z^3 + 28z^2 + 8z + 1$

This mismatching of filter orders can be thought of as system noise, which is represented by N in Figure 12. This noise will act to add misadjustment error to the system which may decrease μ^* from the ideal μ^*.

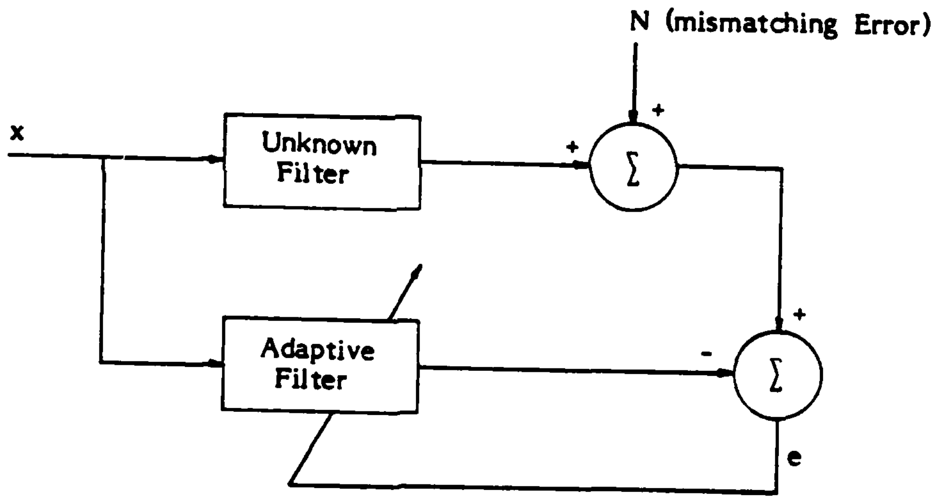


Figure 12
Mismatch Error Represented as Noise

3.0 Simulations

In order to find NXBIT and μ_o which will optimize the filter hardware and to study filter order mismatching error we will find and compare the optimum convergence rate μ^*_{ij} for different combinations of NXBIT (number of bits to which x is rounded) and NU (number of unknown filter weights). The combinations which will be simulated to find μ^*_{ij} are shown in Table 1. Results of these simulations are given in Chapter III.

Table 1.
Optimum Step Size

NXBIT	NU		
	7	8	9
0	μ^*_{07}	μ^*_{08}	μ^*_{09}
1	μ^*_{17}	μ^*_{18}	μ^*_{19}
2	•	•	•
3	•	•	•
4	•	•	•
5	•	•	•
6	μ^*_{67}	μ^*_{68}	μ^*_{69}

CHAPTER III. RESULTS

1.0 Introduction

As discussed in Chapter II, the purpose of this thesis is to define an optimum step size μ_o and NXBIT for optimum convergence to be used in the adaptive filter hardware. The values μ^*_{ij} of Table 1 have been obtained by finding the step size which minimizes the mean square error as discussed in section 2.1 of Chapter II.

In this chapter μ_o and NXBIT are obtained, and the effects of mismatching error are discussed. Also discussed is a better hardware system using sign-magnitude binary numbers as opposed to 2's complement numbers.

2.0 Results

2.1 Determination of Optimum Step Size

We have defined the optimum step size μ^* as the step size which minimizes the mean square error:

$$\frac{\sum_{j=1}^{K} e_j^2}{K}$$

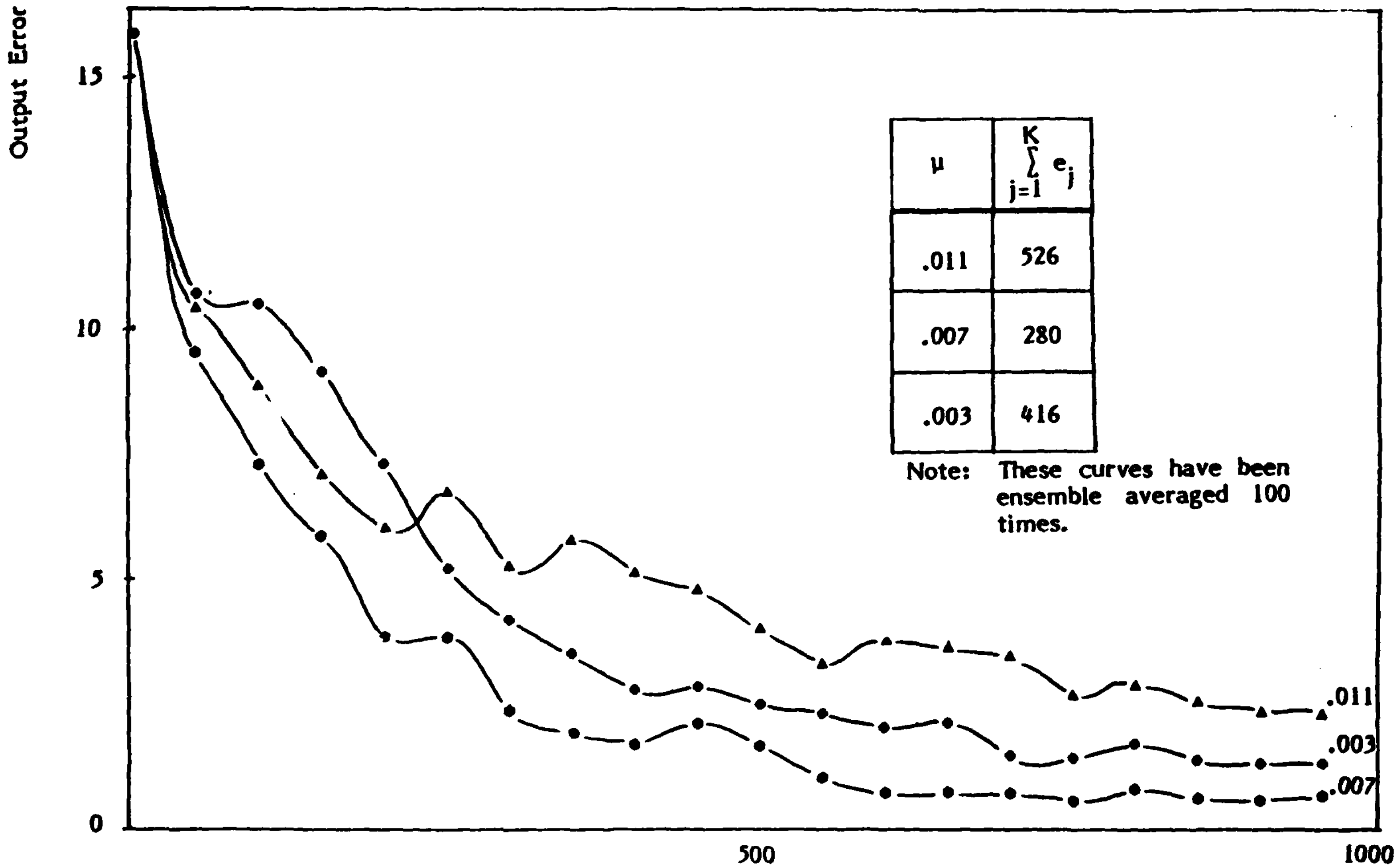
For our case K is constant for all values of μ so that μ^* is the step size which minimizes the total error:

$$\sum_{j=1}^{K} e_j$$

Figure 13 shows the output error curves and total error of different μ values for NXBIT = 3 and NU = 8. The range of step sizes simulated was .003 - .011 with increments of .001, but only three of these were plotted for the sake of clarity. As μ is decreased from .011 the total error decreases until a minimum is reached at μ^*. As μ is decreased from μ^* the total error increases without limit. If total error were plotted as a function of μ the function would have a bowl shape as shown in Figure 14. For the example shown in Figure 13 μ^*_{38} is .007.

2.2 Resulting Optimum Step Sizes

The method discussed in the last section was applied to every combination of NXBIT AND NU to determine all μ^*_{ij}. These values are given in Table 2. The total error for each of the cases is plotted in Figure 15. The output error curves and adaptive filter weight plots are found in Appendix C.



μ	$\sum_{j=1}^{K} e_j$
.011	526
.007	280
.003	416

Figure 13
Output Error and Total Error for Different Values of Step Size

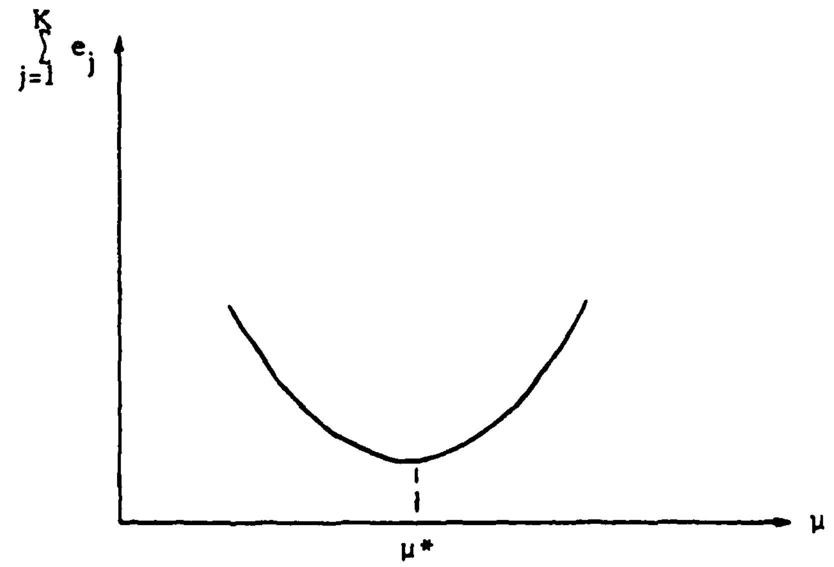


Figure 14
Total Error as a Function of Step Size

Table 2.
Optimum Step Sizes

NXBIT	NU		
	7	8	9
0	.002	.002	.002
1	.005	.005	.005
2	.006	.007	.006
3	.007	.007	.007
4	.007	.007	.008
5	.011	.011	.011
6	.020	.017	.020

From Table 2 and Figure 15 it is clear that the order of the unknown filter does not affect the choice of NXBIT and optimum step size μ_o for the filter hardware. This is a very important result in that it insures that our choice of hardware will work well with unknown systems of varying order.

The value of NXBIT to be used in our hardware will be NXBIT = 3 because the total error of Figure 15 is a minimum for this value. At this value of NXBIT μ^* is .007 so that μ_o will be set at this value. Remember, as shown in section 2.0 of Chapter II, the actual of the filter is scaled by SCALE (130 for our case). The actual step size is then .917.

As was predicted in section 2.2 of Chapter II any effects on the error floor due to the truncation of x are masked by the finite arithmetic errors. This is seen in the error curves in Appendix C. Similarly, as predicted the rate of convergence, which is evaluated by the value of the total error, is affected by the division of bits between e and x. This is seen in Figure 15.

Note: Total error for different values of NU cannot be compared because different weights were used in the unknown filter.

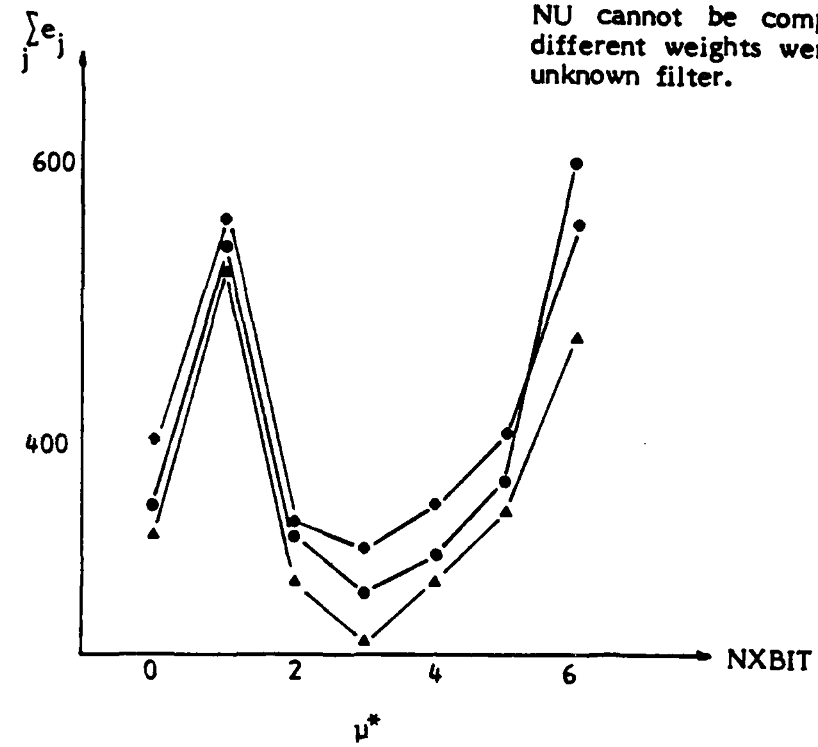


Figure 15
Total Error as a Function of NXBIT

3.0 Sign-Magnitude Binary Filter

In section 1.0 of Chapter II a method was suggested which would improve the accuracy of the adaptive filter hardware. The digital filter as presently implemented uses 2's complement binary numbers. Two's complement arithmetic was developed because design of logic networks to do sign-magnitude arithmetic is awkward. If a system using sign-magnitude numbers can be designed the accuracy of the adaptive filter will be improved.

In 2's complement arithmetic the exclusive OR'ed sign bits of e and x must be fed into the ROM, which calculates $2\mu e' x'$, along with e' and x'. This is because the magnitude of 2's complement numbers are non-distinguishable without their sign bit. An incorrect update quantity would be calculated in the ROM without the sign bits.

In sign-magnitude arithmetic the exclusive OR of the sign bits can post-multiply the update quantity at the output of the ROM. The magnitude of a sign-magnitude number is distinguishable without its sign bit. Because the sign bit need not be fed into the ROM, all 8 ROM inputs are left to divide between e and x. Therefore, the accuracy of either e or x is improved by one bit.

The 8 bits available for e and x can be divided in any manner, just as the 7 bits of the 2's complement system were divided. For the sign-magnitude system further simulations must be run in order to determine a μ_o and NXBIT for optimum convergence to be used in the adaptive filter. The system with NXBIT = 4 is shown in Figure 10b (Chapter II).

CHAPTER IV. CONCLUSION

The purpose of this thesis has been to further develop an adaptive filter hardware and to study the performance and certain characteristics of the filter simulation. Specifically we have determined that NXBIT = 3 and μ_o = .007 are the best values of these two parameters to be used in the hardware. NXBIT is the number of bits to which the input signal x is rounded and μ_o is the optimum step size used in the adaptive updating algorithm. The optimum hardware configuration utilizing these parameter values is shown in Figure 16. It is intended in the future that this update hardware be added to the digital filter hardware presently completed.

It has also been shown that this filter will adapt well to unknown systems of varying order. That is, the order of the unknown filter does not affect the choice of NXBIT and μ_o used in the hardware.

One method of improving the accuracy of the hardware from that shown in Figure 16 is discussed in section 3.0 of Chapter III. Further study is needed to find additional methods of increasing the filter's accuracy.

The results of this thesis have clearly shown that hardware implementation can considerably decrease the accuracy of an ideal adaptive filter. However, the accuracy of this adaptive digital filter is well within the range required for many real systems and should have practical uses in many areas of signal processing.

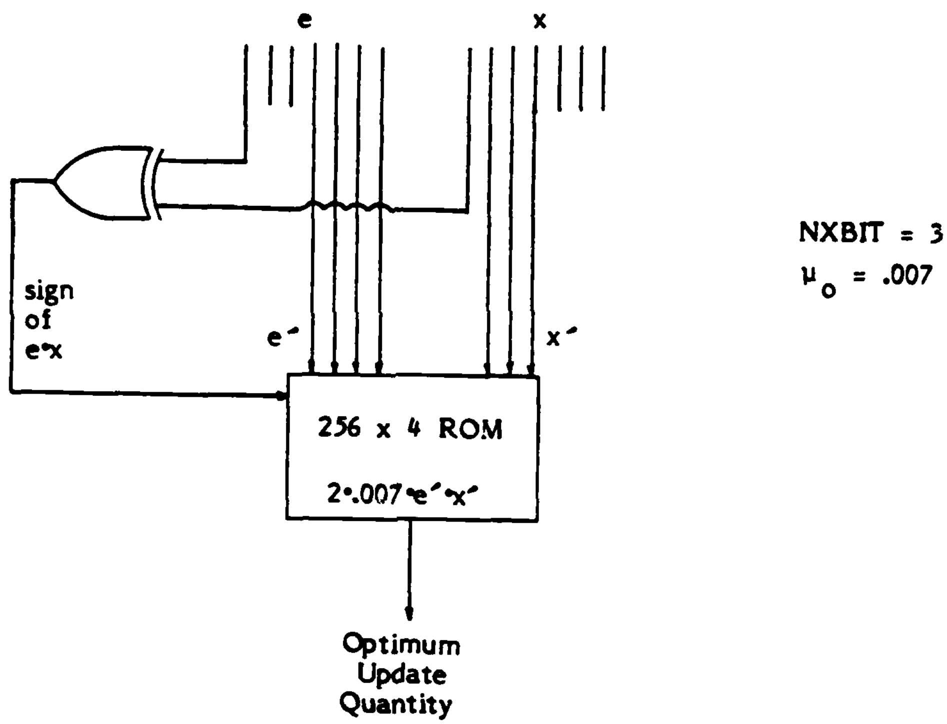


Figure 16
Optimum Hardware Configuration

REFERENCES

1. W.D. Stanley, Digital Signal Processing, Prentice-Hall, 1975.

2. A.V. Oppenheim and R.W. Schafer, Digital Signal Processing, Prentice-Hall, 1975.

3. B. Widrow, J.R. Glouer, Jr., J.M. McCool, J. Kaunitz, C.S. Williams, R.H. Hearn, J.R. Zeidler, E. Dong, Jr., and R.C. Goodlin, "Adaptive Noise Cancellation: Principles and Applications," Proc. IEEE, Vol. 63, No. 12, December 1975, pp. 1692-1716.

4. M.A. Soderstrand, "Cost and Performance Comparisons of Several Implementations of Adaptive Recursive Filters," Proc. 13th Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, CA, November 1979, pp. 416-420.

5. W.A. Gardner and M.A. Soderstrand, "Design and Implementation of Multi-Input Adaptive Signal Extractors," United States Air Force Contract #F49620-79-C-0086.

6. M.A. Soderstrand and J.K. Kelley, "Design of a Low-Cost Adaptive FIR Filter with Sampling Rate in Excess of 10MHz," Proc. IEEE International Symposium on Circuits and Systems, Chicago, Ill., 1981, Vol. 2, pp. 432-434.

7. M.A. Soderstrand, "A High-Speed Low-Cost Recursive Digital Filter Using Residue Number Arithmetic," Proc. IEEE, Vol. 65, No. 7, July 1977, pp. 1065-1067.

8. K.H. O'Keefe, "A Digital Signal Processor Which Uses the Residue Number System," Proc. Mexico 1971 International Conference on Systems, Networks, and Computers, Vol. 2, January 1971, pp. 669-673.

9. W.K. Jenkins and B.J. Leon, "The Use of Residue Coding in the Design of Hardware for Non-recursive Digital Filters," Proc. 8th Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, CA, 1974, pp. 458-462.

10. M.A. Soderstrand, "High-Speed Digital Filters Using Residue Number Arithmetic," Proc. 9th Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, CA, No. 1975, pp. 416-420.

11. M.A. Soderstrand and E.L. Fields, "Multipliers for Residue Number Arithmetic Digital Filters," Electronic Letters, Vol. 13, No. 6, March 17, 1977, pp. 164-166.

12. G.A. Jullien, W.C. Miller, J.J. Soltio, A. Bareniecka, and B. Tseng, "Hardware Realization of Digital Processing Elements Using the Residue Number System," Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, Hartford, CT, May 1977, pp. 506-510.

13. W.K. Jenkins and B.J. Leon, "The Use of RNS in the Design of FIR Digital Filters," IEEE Trans. Circuits and Systems, Vol. CAS-24, No. 4, April 1977, pp. 191-201.

14. A. Baraniecka and G.A. Jullien, "On Decoding Techniques for Residue Number System Realizations of Digital Signal Processing Hardware," IEEE Trans. Circuits and Systems, Vol. CAS-65, No. 11, November 1978, pp. 935-936.

15. N.S. Szabo and R.I. Tanaka, Residue Arithmetic and its Application to Computer Technology, McGraw-Hill, 1967.

16. K.H. O'Keefe, "A Digital Signal Processor Which Uses the Residue Number System," Proc. Mexico 1971 International Conference on Systems, Networks, and Computers, Vol. 2, January 1971, pp. 669-673.

17. M.A. Soderstrand and C. Vernia, "Microprocessor Controlled Development System for Adaptive Filtering Using Parallel Processing and Residue Number Arithmetic," MIMI '80, Montreal, Canada, September 1980.

18. M.A. Soderstrand, "High-Speed Data Conversion Using Residue Number Arithmetic A/D and D/A Converters," Proc. 22nd Midwest Symp. Circuits and Systems, Philadelphia, PA, June 1979. pp. 6-10.

19. W.H. Jenkins, "Techniques for Residue to Analog Conversion for Residue Encoded Digital Filters," IEEE Trans. Circuits and Systems, Vol. CAS-25, No. 7, July 1978. pp. 555-562.

20. F.J. Taylor and A.S. Ramnarayanan, "An Efficient Residue-to-Decimal Converter," T-CAS, December 1981, pp. 1164-1169.

21. M.A. Soderstrand and C. Vernia, "A New Approach to the Implementation of RNS Digital-to-Analog Converters," submitted to Electronics Letters.

22. B. Widrow, J.M. McCool, M.G. Larimore, and C.R. Johnson, Jr., "Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter," Proc. IEEE, Vol. 64, No. 8, August 1976, pp. 1151-1161.

23. T.C. Hsia and S. Sugimoto, "An Investigation of Adjustment Gain Design in Stationary and Nonstationary LMS Adaptive Algorithms," 6th IFAC Symposium on Identification and System Parameter Estimation, Washington, D.C., June 1982.

24. R.D. Gitlin and S.B. Weinstein, "On the Required Tap-weight Precision for Digitally Implemented, Adaptive, Mean-squared Equilizers," The Bell System Tech. J., Vol. 58, February 1979, pp. 301-321.

25. W.A. Gardner, Bounds and Approximations for Learning Characteristics of the Stochastic-Gradient-Descent Vectors, Technical Report No. SIPL-81-11, Digital Signal and Image Processor Laboratory, University of California, Davis, 1981.

APPENDIX A

Program Listing

A listing of the program used to simulate the adaptive filter is included here. As listed, the program plots the absolute value of the difference between the unknown and adaptive filters (error). The adaptive filter weights are also listed, but not plotted.

```
$CONTROL USLINIT,LOCATION,MAP,LABEL
C
          PROGRAM PROJECT
C
C        THIS PROGRAM SIMULATES A PIPELINE RNS ADAPTIVE
C        FILTER, TRYING TO MATCH A USER-ENTERED UNKNOWN
C        FILTER.
C
C*********VARIABLE IDENTIFICATION***********************
C
C         PITT     - NUMBER OF ITERATIONS TO RUN
C         KRUN     - NUMBER OF RUNS TO AVERAGE
C         K        - NUMBER OF MODS IN PIPELINE DESIGN
C         NU       - NUMBER OF WEIGHTS IN UNKNOWN FILTER
C         NA       - NUMBER OF WEIGHTS IN ADAPTIVE FILTER
C         M()      - ARRAY OF MODS FOR PIPELINE RNS FILTER
C         B()      - ARRAY OF CONSTANTS FOR CHI. REM. ALGORITHM
C
C         NXBITS   - NUMBER OF M S BITS TO ROUND X TO
C         NERR     - NUM OF BITS FOR UPPER LIMIT ON ERROR
C         ISL      - FLAG FOR SLOWED-DOWN-UPDATE ALGORITHM
C
C         X        - INPUT SIGNAL
C         JX()     - ARRAY OF DELAYED VALUES OF INPUT SIGNAL AS INTEGER
C                    JX( 1) CONTAINS THE MOST RECENT SIGNAL INPUT
C         IX       - INPUT SIGNAL AS INTEGER
C         Y        - OUTPUT FROM UNKNOWN FILTER
C         IY()     - ARRAY CONTAINING OUPUTS FROM RNS FILTERS
C         YDEC     - OUTPUT FROM RNS FILTERS, CONVERTED TO DECIMAL
C
C         A()      - COEF ARRAY FOR UNKNOWN FILTER (DIM TO NU)
C         Z1()     - DELAY LINES FOR UNKNOWN FILTER
C         Z2()     -
C
C         IA()     - COEF ARRAY FOR ADAPTIVE FILTER (DIM TO K*NA)
C         IZ1()    - ARRAY OF DELAY LINES FOR ADAP FILTER (K*NA)
C         IZ2()    -
C
C         ERR      - DIFFERENCE IN THE TWO FILTER OUTPUTS
C         U        - STEPSIZE FOR UPDATE ALGORITHM
C         IVER     - FLAG FOR UPDATE VERSION DESIRED
C
C         PERRY()  - ARRAY OF STORED ERRORS (EVERY KRUN/100 ITERS)
C         TIM()    - ARRAY FOR X-AXIS OF SUBROUTINE PLOT
C
C         STD      - DESIRED STANDARD DEVIATION FOR RANDOM SIGNAL
C         EX       - DESIRED MEAN VALUE
C         ISD      - DESIRED RANDOM SEED
C
C***********************************************************
C
       DIMENSION IY(4),A(20),Z1(20),Z2(20),IA(4,20),IZ1(4,20)
      C,IZ2(4,20),M(4),B(4),JX(20),PERRY(101),PROD(4)
      C,TIM(101),ICOEF(100,10),COEF(100,10),DIA(20)
       REAL ISD
       DOUBLE PRECISION B,PROD,CHINA
       DATA IY/4*0/,A/20*0.0/,Z1/20*0.0/,Z2/20*0.0/,IA/80*0/,IZ1/80*0/
      C,IZ2/80*0/,M/4*0/,JX/20*0/,PERRY/101*0/,
      CTIM/101*0.0/
C
C     INITIALIZATION BLOCK    (USER INPUT)
C
C
5      WRITE(6,*)'ENTER # OF ITERATIONS TO RUN (1000,5000,ETC):'
       READ(5,*)PITT
       IF (PITT .LT. 5000) VAR=1
       IF (PITT .GE. 5000) VAR=10
7      WRITE(6,*)'NUMBER OF RUNS TO AVERAGE?'
       READ (5,*) KRUN
       IF (KRUN .GT. 0) GOTO 9
         WRITE(6,*)'UNACCEPTABLE PARAMETER!'
         GOTO 7
9      WRITE(6,*) 'NUMBER OF ITERATIONS TO AVERAGE FOR MS ERROR?'
       READ(5,*) KMSE
       WRITE(6,*)'HOW MANY WEIGHTS IN THE UNKNOWN FILTER?'
```

```
      READ (5,*) NU
      WRITE(6,*) 'HOW MANY WEIGHTS IN THE ADAPTIVE FILTER?'
      READ(5,*) NA
16    WRITE(6,*)'ENTER NUMBER OF M S BITS TO ROUND X TO: 2,3,4,5.'
      WRITE(6,*)'ENTER 0 IF X IS NOT TO BE ROUNDED.'
      READ(5,*)NXBIT
      IF (NXBIT .GE. 0 .AND. NXBIT .LE. 6) GOTO 17
        WRITE(6,*)'UNACCEPTABLE PARAMETER!'
        GOTO 16
17    WRITE(6,*)'ENTER NUMBER OF BITS FOR UPPER BOUND ON ERROR'
      WRITE(6,*)'SHOULD BE 7 - # OF BITS FOR X.'
      READ(5,*)NERR
      IF (NERR .GE. 0 .AND. NERR .LE. 7) GOTO 18
        WRITE(6,*)'UNACCEPTABLE PARAMETER!'
        GOTO 17
C
18    K=4
      SERR=0.
      DO 20 I=1,101
20          TIM(I)=FLOAT(I-1)*(PITT/100)
C
C
      DO 22 L=1,100
      DO 22 I=1,NA
      COEF(L,I)=0
22    CONTINUE
C
      DO 30 II=1,KRUN
      CALL MAIN(X,IX,Y,IY,A,Z1,Z2,IA,IZ1,IZ2,K,NU,NA,B,JX,
     CPERRY,KRUN,PITT,VAR,II,NXBIT,NERR,ISL,PROD,
     CII,U,M,SCALE,STD,EX,ISD,ICOEF)
      DO 30 L=1,100
      DO 30 I=1,NA
      COEF(L,I)= COEF(L,I) +FLOAT(ICOEF(L,I))/KRUN
30    CONTINUE
      WRITE (2,*)
      WRITE (2,*)'ADAPTIVE WEIGHTS GIVEN EVERY',
     CPITT/100.,'ITERATIONS'
      DO 31 L=1,100
      WRITE (2,*) (IFIX(COEF(L,I)),I=1,NA)
31    CONTINUE
C
C
      WRITE(2,*)
      WRITE(2,*)' FINAL ADAPTIVE COEFFICIENTS ARE:'
      WRITE(6,*)' FINAL ADAPTIVE COEFFICIENTS ARE:'
      DO 90 J=1,K
      WRITE(2,80) M(J),(IA(J,I),I=1,NA)
      WRITE(6,60) M(J),(IA(J,I),I=1,NA)
60    FORMAT(' MOD',I3,':',10I6)
90    CONTINUE
      CALL DECIML(NA,K,IA,DIA,6,M,PROD)
      WRITE (2,*)
      WRITE (6,*)
      WRITE (2,*) (IFIX(DIA(I)),I=1,NA)
      WRITE (6,*) (IFIX(DIA(I)),I=1,NA)
      WRITE(2,*)
      WRITE(6,*)
      DO 95 I=1,101
          PERRY(I)=PERRY(I)/FLOAT(KRUN)
      IF((I-((KMSE/(IFIX(PITT/100.)))+1))) 94,95,95
94    SERR=SERR +PERRY(I)
95    CONTINUE
      WRITE(6,*)
      WRITE(2,*)
      WRITE(2,*) 'SUM OF ERRORS=', SERR
      WRITE(6,*) 'SUM OF ERRORS=', SERR
      WRITE(2,*)
      WRITE(2,*)' ERROR VALUES GIVEN EVERY',PITT/100.,
     C'ITERATIONS'
      WRITE(2,*)PERRY
      CALL PLOT(TIM,PERRY,101,0)
900   STOP
      END
C
C*******************************************************************
C***********MAIN SUBROUTINE*****************************************
C*******************************************************************
C
      SUBROUTINE MAIN(X,IX,Y,IY,A,Z1,Z2,IA,IZ1,IZ2,K,NU,NA,B,
```

```
     CJX,PERRY,KRUN,PITT,VAR,II,NXBIT,NERR,ISL,PROD,
     CI1,U,M,SCALE,STD,EX,ISD,ICOEF)
C
C      THIS SUBROUTINE SIMULATES THE TWO FILTERS AND COMPARES
C      OUTPUTS
C
      DIMENSION IY(K),A(NU),Z1(NU),Z2(NU),IA(K,NA),IZ1(K,NA),
     CIZ2(K,NA), M(K),B(K),IDX(K,NA),JX(NA),PERRY(101),PROD(K)
     C,IW(K),DIA(NA),ICOEF(100,NA)
      REAL ISD
      DOUBLE PRECISION B,PROD,CHINA
C
C
      CALL INIT(IY,IA,IZ1,IZ2,K,NU,NA,JX,II)
C
C      ************************
C      * INITIALIZATION BLOCK *
C      ************************
C
      IF (II .GT. 1) GOTO 40
C
C* OPTIONS FOR FILLING ERROR ARRAY
C
      WRITE(6,*)'AVERAGE THE ERROR VALUES OR JUST SAMPLE?'
      WRITE(6,*)'TO JUST SAMPLE, PRESS 1'
      READ (5,*) I1
C
C* OPTIONS FOR UPDATING (SLOWED-DOWN ...)
C
8     WRITE(6,*)'SLOWED-DOWN UPDATING OR UPDATE EVERY ITERATION?'
      WRITE(6,*)'TO UPDATE EVERY ITERATION, HIT 1,'
      WRITE(6,*)'TO SLOW DOWN, HIT 2.'
      READ(5,*)ISL
      IF (ISL .EQ. 1 .OR. ISL .EQ. 2) GOTO 9
        WRITE(6,*)'UNACCEPTABLE PARAMETER!'
        GOTO 8
C
C* INPUT COEFFICIENTS OF UNKNOWN SYSTEM
C
9     DO 10 I=1,NU
          WRITE(6,*)'VALUE FOR A(',I,')?'
          READ(5,*)A(I)
10    CONTINUE
C
C* INITIALIZE MODS
C
      M(1)=11
      M(2)=13
      M(3)=15
      M(4)=16
      DO 20 J=1,K
          DO 20 L=1,NA
            IA(J,L)=MODP(IA(J,L),M(J))
20    CONTINUE
      WRITE(6,*)'ENTER MU FOR UPDATE ALGORITHM:'
      READ(5,*) U
      WRITE(6,*)'ENTER SCALING FACTOR FOR RNS FILTER INPUT:'
      READ(5,*)SCALE
C
C* INPUT INFORMATION FOR RANDOM NUMBER GENERATOR
C
      WRITE(6,*)
      WRITE(6,*)'DESIRED STANDARD DEVIATION?'
      READ(5,*)STD
      WRITE(6,*)'DESIRED MEAN VALUE?'
      READ(5,*)EX
      WRITE(6,*)'DESIRED RANDOM SEED?'
      READ(5,*)ISD
C
C* INPUT VALIDATION - ECHO CHECK
C
      WRITE(2,*)'***************************************'
      WRITE(2,*)'RNS PIPELINE ADAPTIVE FILTER SIMULATION'
      WRITE(2,*)
      WRITE(2,*)' *----*-   PARAMETERS INITIALIZED AS FOLLOWS:'
      WRITE(2,*)
      WRITE(2,799) KRUN,PITT
799    FORMAT(X,I4,' RUNS OF' ,F7.0, ' ITERATIONS EACH.',/)
      WRITE(2,800)K,(M(I),I=1,K)
      WRITE(6,800)K,(M(I),I=1,K)
```

```
      WRITE(2,801)(A(I),I=1,NU)
      WRITE(6,801)(A(I),I=1,NU)
801   FORMAT(' UNKNOWN FILTER COEFFS INITIALIZED TO:',/,10F8.5/)
      WRITE(2,802)STD
      WRITE(6,802)STD
802   FORMAT(/,' RANDOM SIGNAL PARAMETERS:',/,' STAN DEVIATION=',F5.3)
      WRITE(2,803)EX,ISD
      WRITE(6,803)EX,ISD
803   FORMAT(' MEAN VALUE=',F3.0,/,' RANDOM SEED=',F10.0/)
      WRITE(2,*)
      WRITE(2,804)U,NXBIT,NERR
804   FORMAT(' FOR UPDATING:',/,'    MU=',F8.5,/,
     1       '    NUM OF BITS X ROUNDED TO:',I3,/,
     2       '    NUM OF BITS ALLOWED FOR ERROR:',I3,/)
      IF (ISL .EQ. 2) WRITE(2,*)'NOTE:SLOWED-DOWN-UPDATING!'
      IF (ISL .EQ. 1) WRITE(2,*)'NOTE:UPDATED EVERY ITERATION.'
      WRITE(2,*)
      WRITE(2,*)'***SCALING FACTOR FOR RNS FILTER',SCALE,'***'
      WRITE(2,*)
C
C      ****************************
C      * BEGIN FILTER SIMULATION *
C      ****************************
C
      CALL WEIGHT (M,B,K,PROD)
40    ERR = 0.0
      ICNTR=0
      DO 41 L=1,100
      DO 41 I=1,NA
       ICOEF(L,I)=0
41    CONTINUE
      ITER=1
      SUMM=0
      J1=IFIX(PITT/(100.*VAR))
      IX0=0
      IX1=0
      IX2=0
      DO 200 L=1,100
        DO 100 LP1=1,IFIX(PITT/100.)
C
C* OBTAIN AND SCALE INPUT SIGNAL
C
      CALL NORMAL (STD,EX,ISD,X)
      IX=X*SCALE
C
C* ROUND IX TO M S BITS => THIS VERSION OF IX GOES TO UPDATE ALGORITHM
C                           BUT ORIGINAL IX IS PASSED THRU ADAP FLTR
C
      IXX=IX
      IX=IROUND(IX,NXBIT)
101   IX0=IX2
      IX2=IX
C
C* UPDATE DELAY BLOCK JX
C       JX(1) CONTAINS THE MOST RECENT SIGNAL INPUT
C
      NAM1=NA-1
      DO 102 I=1,NAM1
        JX(NA+1-I)=JX(NA-I)
102   CONTINUE
      JX(1)=IX0
C
C* PASS X THRU UNKNOWN FILTER
C
      CALL UNKNOW(X,A,Z1,Z2,Y,NU)
C
C* PASS X THRU ADAPTIVE FILTER
C
      CALL ADPTIV(IXX,IA,IZ1,IZ2,M,NA,IY,K)
C
C* CONVERT ADAPTIVE RNS OUTPUT TO DECIMAL, THEN
C        CALCULATE ERROR BETWEEN ADAPTIVE FILTER AND
C        UNKNOWN FILTER
C
      FCHINA=CHINA(IY,B,K,M,PROD)
      YDEC=FCHINA/SCALE
      ERR=(Y*SCALE)-YDEC
C
C* UPDATE ERROR ARRAY
```

```
      C
            ABSERR=ABS(ERR)
              SUMM=SUMM+ABSERR
               IF (LP1 .NE. IFIX(PITT/100.)) GOTO 105
                  ITEN=ITEN+1
                  IF (II .EQ. 1) GOTO 104
                  PERRY(ITEN)=(100.*SUMM)/PITT+PERRY(ITEN)
      104         IF (II .EQ. 1) PERRY(ITEN)=ABSERR+PERRY(ITEN)
                  SUMM=0
      105   CONTINUE
      C
      C* CHANGE OUTPUT FROM UNKNOWN INTO EACH OF THE GIVEN MODS
      C          COMPARE TO RNS OUTPUTS (COMPARE RESPECTIVE MODS)
      C          AND DETERMINE APPROPRIATE STEPSIZES, AND UPDATE)
      C
      C
            ICNTR=ICNTR+1
            LMOD=MOD(ICNTR,6)
            IF (ISL .EQ. 2) GOTO 108
               ERR1=ERR
               GOTO 110
      108   IF(LMOD.LT.1) GOTO 100
            IF(LMOD.GT.6) GOTO 100
            IF(LMOD.NE.1) GOTO 110
            ERR1=ERR
      C
      110   CALL UPDATE(JX,U,Y,IY,NA,ERR1,IA,M,K,LMOD,NERR,ISL)
      C
      100   CONTINUE
            CALL DECIML(NA,K,IA,DIA,B,M,PROD)
            DO 92 I=1,NA
            ICOEF(L,I)=DIA(I)
      92    CONTINUE
      200   CONTINUE
            RETURN
            END
      C
      C     *************************
      C     * INITIALIZE SUBROUTINE *
      C     *************************
      C
      C   THIS SUBROUTINE INITIALIZES SOME ARRAYS TO ZERO
      C
            SUBROUTINE INIT(IY,IA,IZ1,IZ2,K,NU,NA,JX,II)
            DIMENSION IY(K),IZ1(K,NA),IZ2(K,NA),IA(K,NA),JX(NA)
            DO 10 I=1,K
               IY(I)=0
               DO 10 L=1,NA
                  IA(I,L)=0
                  IZ1(I,L)=0
                  IZ2(I,L)=0
                  JX(L)=0
      10     CONTINUE
            IF (II .GT. 1) GOTO 40
            DO 20 L=1,NA
            WRITE(6,*)'INIT VALUE FOR IA(1,',L,')'
            READ(5,*)IA(1,L)
      20    CONTINUE
            DO 30 I=2,K
               DO 30 L=1,NA
                  IA(I,L)=IA(1,L)
      30    CONTINUE
      40    RETURN
            END
      C
      C     ******************
      C     * CHINA FUNCTION *
      C     ******************
      C
            FUNCTION CHINA(IY,B,K,M,PROD)
            DIMENSION IY(K),B(K),M(K),PROD(K)
            DOUBLE PRECISION CHINA,B,PROD,PRODM
            CHINA = 0
            DO 10 I=1,K
               CHINA=CHINA+B(I)*IY(I)*PROD(I)
      10    CONTINUE
            PRODM=M(1)*PROD(1)
            CHINA=CHINA-DDINT(CHINA/PRODM)*PRODM
            IF ((CHINA/PRODM) .GE. 0.5) CHINA=CHINA-PRODM
            RETURN
```

```
      END
C
C
C     *********************
C     * IROUND FUNCTION *
C     *********************
C
      FUNCTION IROUND (IX,NXBIT)
C
       IF (NXBIT.NE.0) GOTO 10
       IROUND = ISIGN(128,IX)
       RETURN
 10    LX = 2**(7-NXBIT)
       IROUND = LX*(IX/LX)
       RETURN
       END
C
C
C     **************************
C     * POSITIVE MOD FUNCTION *
C     **************************
C
      FUNCTION MODP(I,M)
      IF (MOD(I,M).GE.0)GOTO 10
      MODP=M+MOD(I,M)
      RETURN
   10 MODP=MOD(I,M)
      RETURN
      END
C
C
C     **********************
C     * CONVERT TO DECIMAL*
C     **********************
C
      SUBROUTINE DECIML (NA,K,IA,DIA,B,M,PROD)
      DIMENSION IA(K,NA),DIA(NA),B(K),M(K),PROD(K),IW(K)
      DOUBLE PRECISION B,PROD,CHINA
      DO 10 I=1,NA
      DO 20 J=1,K
      IW(J)=IA(J,I)
 20   CONTINUE
      DIA(I)=CHINA(IW,B,K,M,PROD)
 10   CONTINUE
      RETURN
      END
C
C
C
C     **********************
C     * UPDATE SUBROUTINE *
C     **********************
C
      SUBROUTINE UPDATE(JX,U,Y,IY,NA,ERR,IA,M,K,LMOD,NERR,ISL)
      DIMENSION JX(NA),M(K),IY(K),IA(K,NA)
      DIMENSION KK(K,NA)
C
C           THERE IS ONLY ONE DELAY LINE OF THE INPUT SIGNAL.
C           UPDATE ALGORITHM USES THESE DELAYED VALUES, USING
C           MODULAR ARITHMETIC.
C
C   ****VARIABLE IDENTIFICATION:*******************************
C   *                                                         *
C   *            TU : 2*U                                     *
C   *            KY : OUTPUT Y FROM UNKNOWN, AS INTEGER       *
C   *         ISTEP : SIZE OF STEP, EITHER M(1) OR            *
C   *                   ERROR BETWEEN THE TWO FILTERS         *
C   *           ISL : FLAG FOR SLOWED-DOWN-UPDATE ALGORITHM   *
C   *                                                         *
C   ***********************************************************
C
C     IN ADDITION, THE UPDATE ALGORITHM USES TWO DIFFERENT
C     STEPSIZE INCREMENTS, DEPENDING ON WHETHER OR NOT THE
C     ERROR IS LESS THAN THE SMALLEST MOD.
C
       ISTEP=ERR
C
C* IF ABS(ERR)>2**NERR-1, ERROR IS FIXED AT SAME
C
      IF (IABS(ISTEP) .LE. 2**NERR-1) GOTO 15
      ISTEP=2**NERR-1
      IF (ERR .LT. 0) ISTEP=-ISTEP
 15   IF (ISL .EQ. 1) GOTO 50
```

```
C
C**UPDATE ONE COEFFICIENT ONLY ON COUNTS 1 - 8
C
20    ITEMP=2*(U*ISTEP)*JX(NA)
      NAML=NA+1-LMOD
      IF ((NAML.LE.0).OR.(NAML.GT.NA)) WRITE(6,*)'BOUNDS TROUBLE!'
      DO 30 J=1,K
        KTEMP=IA(J,NAML)+ITEMP
        IA(J,NAML)=MODP(KTEMP,M(J))
C
30    CONTINUE
40    RETURN
C
C* UPDATE EVERY ITERATION
C
50    DO 60 I=1,NA
         ITEMP=2*(U*ISTEP)*JX(I)
         DO 60 J=1,K
            KTEMP=IA(J,I)+ITEMP
            IA(J,I)=MODP(KTEMP,M(J))
60    CONTINUE
      RETURN
      END
C
C
C
C
C
C     ******************************
C     * UNKNOWN FILTER SUBROUTINE *
C     ******************************
C*****************-----------------------------------------------**********
C*   DUMMY VARIABLE DECLARATION:                                         *
C*       X     :   INPUT                                                 *
C*       A()   :   ARRAY OF COEFFICIENTS                                 *
C*       Z1()  :   DELAY ARRAY I                                         *
C*       Z2()  :   DELAY ARRAY II                                        *
C*       Y     :   OUTPUT                                                *
C*       N     :   NUMBER OF WEIGHTS                                     *
C************************************************************************
      SUBROUTINE UNKNOW(X,A,Z1,Z2,Y,N)
      DIMENSION A(N),Z1(N),Z2(N)
C
C* UPDATE Z2 FIRST
C
      NM1=N-1
      DO 10 I=1,NM1
         Z2(I)=Z1(I)+Z2(I+1)
   10 CONTINUE
      Z2(N)=Z1(N)
C
C* UPDATE Z1
C
      DO 20 I=1,N
         Z1(I)=X*A(I)
   20 CONTINUE
      Y=Z2(1)
      RETURN
      END
C
C     ********************************
C     * ADAPTIVE FILTER SUBROUTINE *
C     ********************************
C
      SUBROUTINE ADPTIV(IX,IA,IZ1,IZ2,M,NA,IY,K)
      DIMENSION IA(K,NA),IZ1(K,NA),IZ2(K,NA),M(K),IY(K)
C
C* UPDATE IZ2 FIRST
C
      DO 50 J=1,K
      NM1=NA-1
      DO 10 I=1,NM1
         IZ2(J,I)=MODP(IZ1(J,I)+IZ2(J,I+1),M(J))
   10 CONTINUE
      IZ2(J,NA)=IZ1(J,NA)
C
C* UPDATE IZ1
      DO 20 I=1,NA
C
         IZ1(J,I)=MODP(IX*IA(J,I),M(J))
```

```
   20 CONTINUE
          IY(J)=IZ2(J,1)
50     CONTINUE
       RETURN
       END
C
C
C
C      ***************************
       SUBROUTINE WEIGHT (M,B,K,PROD)
C     ***************************
C
       DIMENSION M(K),B(K),PROD(K)
       DOUBLE PRECISION B,PROD
       REAL MF
       DO 100 I=1,K
          MF=1
          DO 10 J=1,K
             IF (J .EQ. I) GOTO 10
             MF=MF*M(J)
10        CONTINUE
           PROD(I)=MF
          MF=AMOD(MF,FLOAT(M(I)))
          JPOINT=M(I)-1
          DO 20 J=1,JPOINT
             NSAVE=J
             ITEMP=IFIX(AMOD(MF*J,FLOAT(M(I))))
             IF (ITEMP .EQ. 1) GOTO 30
20        CONTINUE
          WRITE (6,25)
25        FORMAT (X,' ERROR 25')
C
30        B(I)=FLOAT(NSAVE)
100    CONTINUE
       RETURN
       END
C
C
C
C
C ************************************************************
C SUBROUTINE TO GENERATE A NORMALLY DISTRIBUTED RANDOM VAR
C VARIABLES:
C       EX     = DESIRED MEAN VALUE
C       STD    = DESIRED STANDARD DEVIATION
C       ISEED1 = SEQUENCE STARTING SEED1
C       RV1    = RETURNED RANDOM NUMBER
C ************************************************************
       SUBROUTINE NORMAL (STD,EX,ISEED1,RV1)
       REAL ISEED1
C ------------------------------------------------------------
C .....IF STD IS 0, THEN DO NO COMPUTATIONS
       IF (STD.EQ.0.0) GOTO 20
C .....GENERATE RANDOM #S AND SCALE TO LIE WITHIN (-1,1)
C
C      WRITE(6,*)'2.X=',RV1,'STD=',STD,'EX=',EX,'ISD=',ISEED1
10     X1=-1.0+2.0*RAND(ISEED1)
C      WRITE(6,*)'AT 411.1, EX=',EX
       X2=-1.0+2.0*RAND(ISEED1)
C      WRITE(6,*)'3.X=',RV1,'STD=',STD,'EX=',EX,'ISD=',ISEED1
C
C .....FIND THE NORM SQUARED OF (X1,X2)
C
       S=X1*X1+X2*X2
C      WRITE(6,*)'AT 416.1, EX=',EX
C
C .....CHECK TO SEE IF (X1,X2) LIES WITHIN UNIT CIRCLE
C
       IF (S.GE.1.0)GOTO 10
C      WRITE(6,*)'AT 420.1, EX=',EX,'X1=',X1,'X2=',X2,'S=',S
C
C .....IF S=0 THEN GENERATE APPROPRIATE NORMAL RANDOM NUMBERS
C
       IF(S.EQ.0.0)GOTO 20
C      WRITE(6,*)'AT 424.1, EX=',EX
C
C .....GENERATE NORMAL RANDOM NUMBERS
C
       W=SQRT((-2.0*ALOG(S))/S)
C      WRITE(6,*)'AT 428.1, EX=',EX
```

```
      RV1=EX+STD*W*X1
C     WRITE(6,*)'AT 429.1, EX=',EX
      RETURN
C
20    RV1=EX
      RETURN
      END
C
C
C
C *************************************************************
C * PLOT IS A SUBROUTINE FOR PRINTING A 2-D GRAPH OF 2 VARS   *
C * --VARIABLES------------------------------------------------*
C *     X:       A SINGLE SUBSCRIPTED REAL ARRAY DIMENSIONED TO *
C *              NPTS IN MAIN. X FORMS THE AXIS OF THE ABSCISSAS *
C *              OF THE GRAPH.                                   *
C *     Y:       A SINGLE SUBSCRIPTED REAL ARRAY DIMENSIONED TO  *
C *              NPLOT IN MAIN. Y FORMS THE AXIS OF ORDINATES OF *
C *              THE GRAPH.                                      *
C *     NPTS:    NUMBER OF POINTS TO BE PLOTTED FOR EACH PLOT ON *
C *              THE GRAPH.                                      *
C *     IT:      DETERMINES WHETHER X AXIS IS LINEAR OR LOG      *
C *              IT=1 => LOG SCALE                               *
C *              IT NOT 1 => LINEAR SCALE                        *
C *************************************************************
      SUBROUTINE PLOT (X,Y,NPTS,IT)
      REAL X(NPTS),Y(NPTS),XSCAL(11)
      LOGICAL PT
      CHARACTER BL,BLANK,DIV,BAR,MINUS,PLUS,LINE(101),SYMB(2)
      DATA BLANK/' '/,BAR/'.'/,MINUS/'-'/,PLUS/'+'/,SYMB/'X','O'/
C
C.....CHECK TO SEE IF #PLOTS DESIRED IS 1 OR 2
C
C
C.....INITIAL DATA SET UP---FIND MAX VALUES OF X AND Y
C
10    YMAX=Y(1)
      YMIN=YMAX
      DO 20 J=1,NPTS
         YT=Y(J)
         IF (YT.GT.YMAX) YMAX=YT
         IF (YT.LT.YMIN) YMIN=YT
20    CONTINUE
      XMAX=X(1)
      XMIN=XMAX
      DO 30 I=1,NPTS
         XT=X(I)
         IF (XT.GT.XMAX) XMAX=XT
         IF (XT.LT.XMIN) XMIN=XT
30    CONTINUE
C
C.....FIND THE RANGE OF Y AND X VALUES
C
      YR=YMAX-YMIN
      XR=XMAX-XMIN
C
C.....CHECK TO SEE IF LOG SCALE (IT=1 => LOG SCALE)
C
      IF (IT.NE.1) GOTO 50
C
C.....CALCULATE SCALING FACTOR NEEDED TO PLACE POINTS ON LINE
C
      DFL=(ALOG10(XMAX)-ALOG10(XMIN))/100.00
C
C.....CALCULATE SCALING FACTOR NEEDED TO PLACE SCALE ON X-AXIS
C
      DF=10.**((ALOG10(XMAX)-ALOG10(XMIN))/10.00)
C
C.....SET UP EACH LINE OF GRAPH (HORIZONTAL GRID)
C
50    IYS=0
      WRITE(2,900)
      DO 130 IYT=1,51
         IYA=52-IYT
         IYS=IYS+1
         BL=BLANK
         DIV=BAR
         PT=.FALSE.
         IF (IYS.NE.1) GOTO 60
         BL=MINUS
```

```
                DIV=PLUS
                PT=.TRUE.
                YSCAL=YMAX-(IYI-1)*YR/50.
      C
      C.....SET UP VERTICAL GRID
      C
      60        DO 70 IXA=1,101
      70           LINE(IXA)=BL
                DO 80 IXA=1,101,10
      80           LINE(IXA)=DIV
      C
      C.....INSERT DATA POINTS ALONG GRAPH LINE
      C
                DO 110 M=1,NPTS
                   IY=50.*(Y(M)-YMIN)/YR+1.4999
                   IF (IT.NE.1) GOTO 90
                   IX=(ALOG10(X(M))-ALOG10(XMIN))/DFL+1.4999
                   GOTO 100
      90           IX=100.*(X(M)-XMIN)/XR+1.4999
      100          IF (IY.NE.IYA) GOTO 110
                   LINE(IX)=SYMB(1)
      110       CONTINUE
      C
      C.....PRINT Y-AXIS VALUE EVERY FIFTH ROW IF PT IS TRUE
      C
                IF (PT) WRITE(2,1000)YSCAL,LINE
                IF (.NOT.PT) WRITE(2,1100)LINE
                IF (IYS.EQ.5) IYS=0
      130   CONTINUE
      C
      C.....PRINT X-AXIS SCALE VALUES
      C
            DO 150 IXM=1,11
               IF (IT.NE.1) GOTO 140
               XSCAL(IXM)=XMIN*DF**(IXM-1)
               GOTO 150
      140      XSCAL(IXM)=XMIN+(IXM-1)*XR/10.
      C***************************************************************
      150   CONTINUE
            WRITE(2,1200)XSCAL(1),XSCAL(3),XSCAL(5),XSCAL(7),XSCAL(9),
           CXSCAL(11),XSCAL(2),XSCAL(4),XSCAL(6),XSCAL(8),XSCAL(10)
      900   FORMAT(1H1,"PLOT OF THE ERROR")
      1000  FORMAT(' ',5X,1G11.4,1X,101A1)
      1100  FORMAT(' ',17X,101A1)
      1200  FORMAT('0',3X,6G20.3/13X,5G20.3)
      500   RETURN
            END
```

APPENDIX B

Simulated System

The system simulated by the program in Appendix A is shown in Figure 17. The signal is scaled by SCALE at several points in the system because the Residue Number System requires integer arithmetic. We would like SCALE to be as large as possible without causing the system to exceed the RNS numbers available. Because we have chosen the mods 11, 13, 15 and 16 the range of integers available is $(-1/2(11 \cdot 13 \cdot 15 \cdot 16), 1/2(11 \cdot 13 \cdot 15 \cdot 16))$. To calculate the value of SCALE to use we will assume the system is a pipelined TDL with the weights w_j normalized such that $\sum_{i=o}^{n} w_i$ = SCALE. The output of the filter is

$$y_i = \sum_{i=o}^{n} w_i \; x(i-2)$$

which is less than

$$|x_{max}| \cdot \sum_{i=o}^{n} w_i$$

If the input signal is assumed to be maximum at the value SCALE the output becomes

$$y_{max} = (\mathrm{SCALE}^2)$$

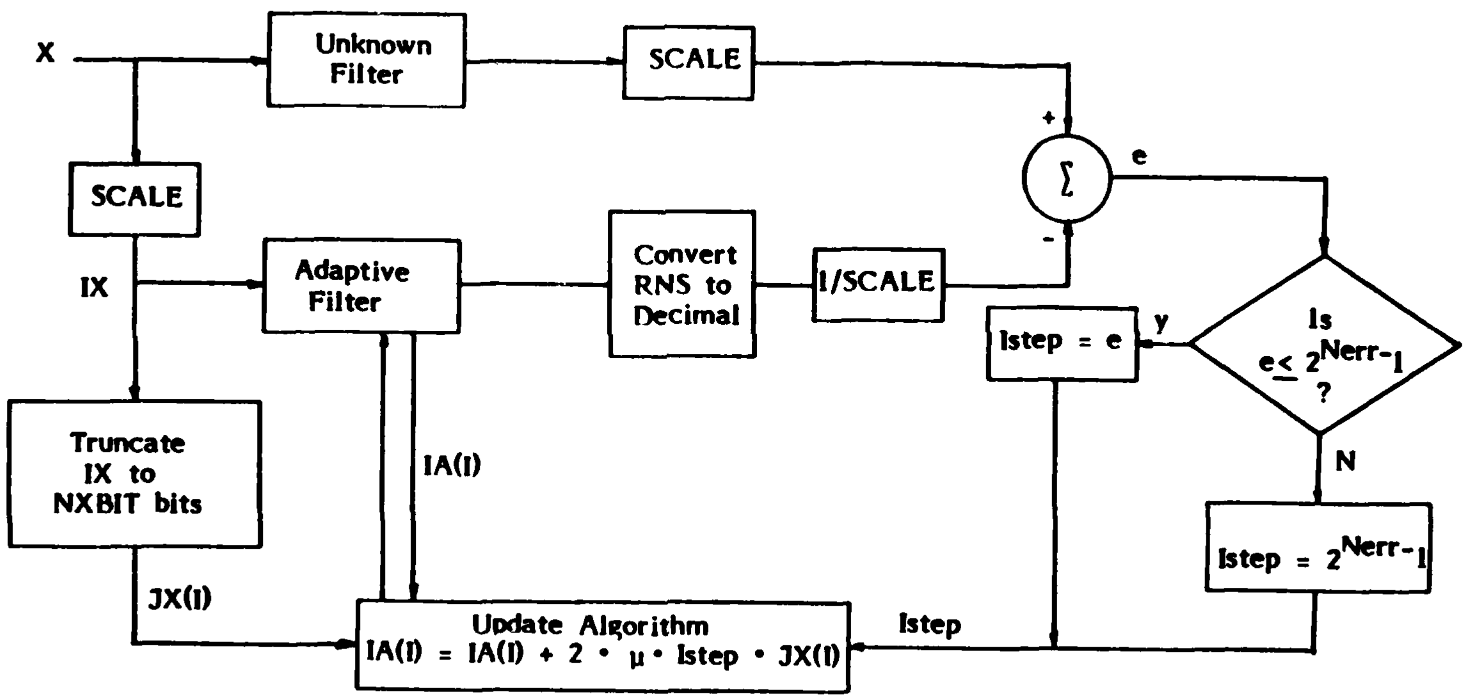
which must not exceed the range of RNS numbers, therefore:

$$y_{max} = 1/2(11 \cdot 13 \cdot 15 \cdot 16)$$

$$\mathrm{SCALE}^2 = 1/2(11 \cdot 13 \cdot 15 \cdot 16)$$

The value we will use in the adaptive filter will then be

$$\mathrm{SCALE} = 130.996.$$



NERR = 7-NXBIT

Figure 17
Simulated System Diagram

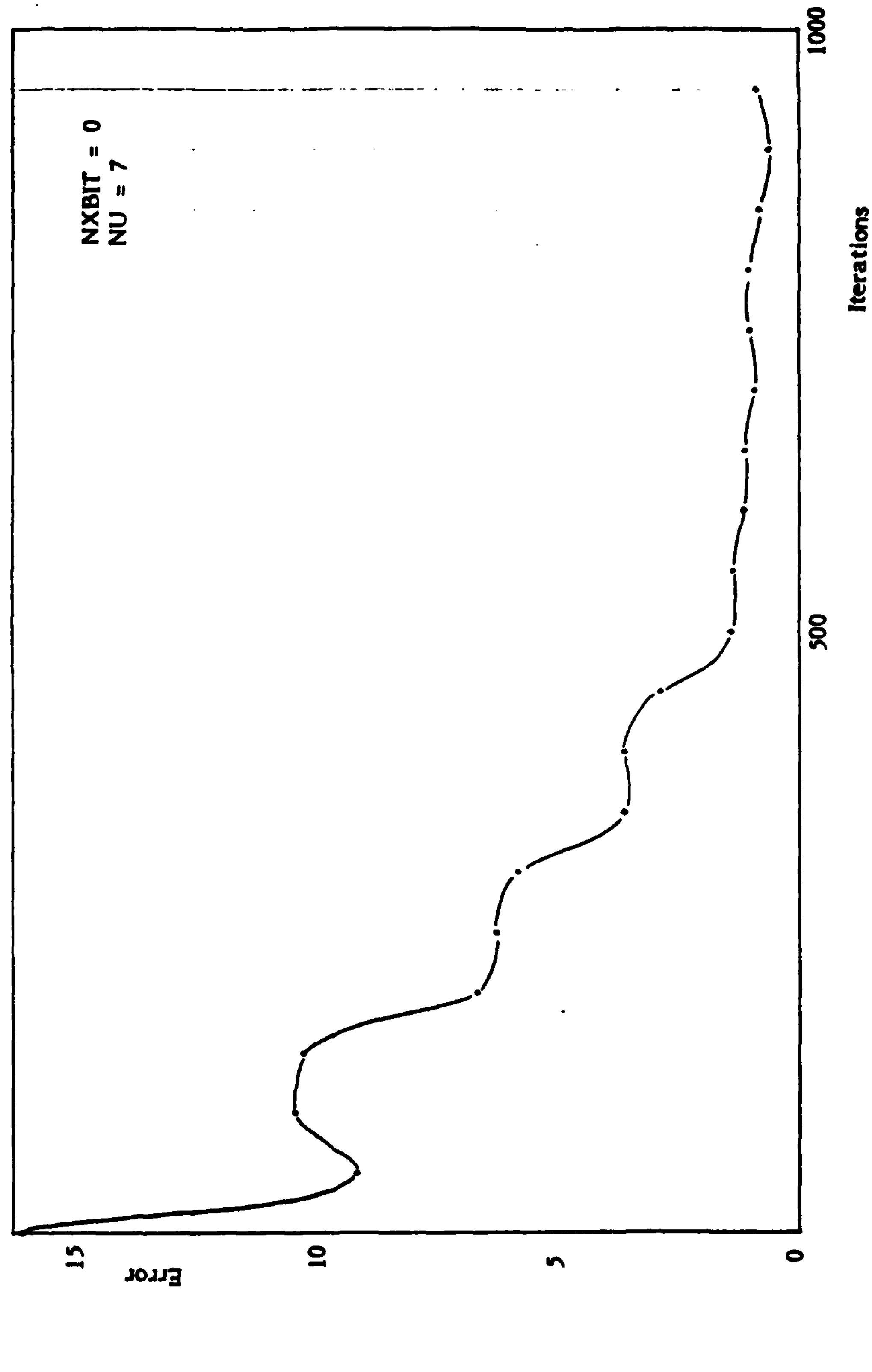
APPENDIX C

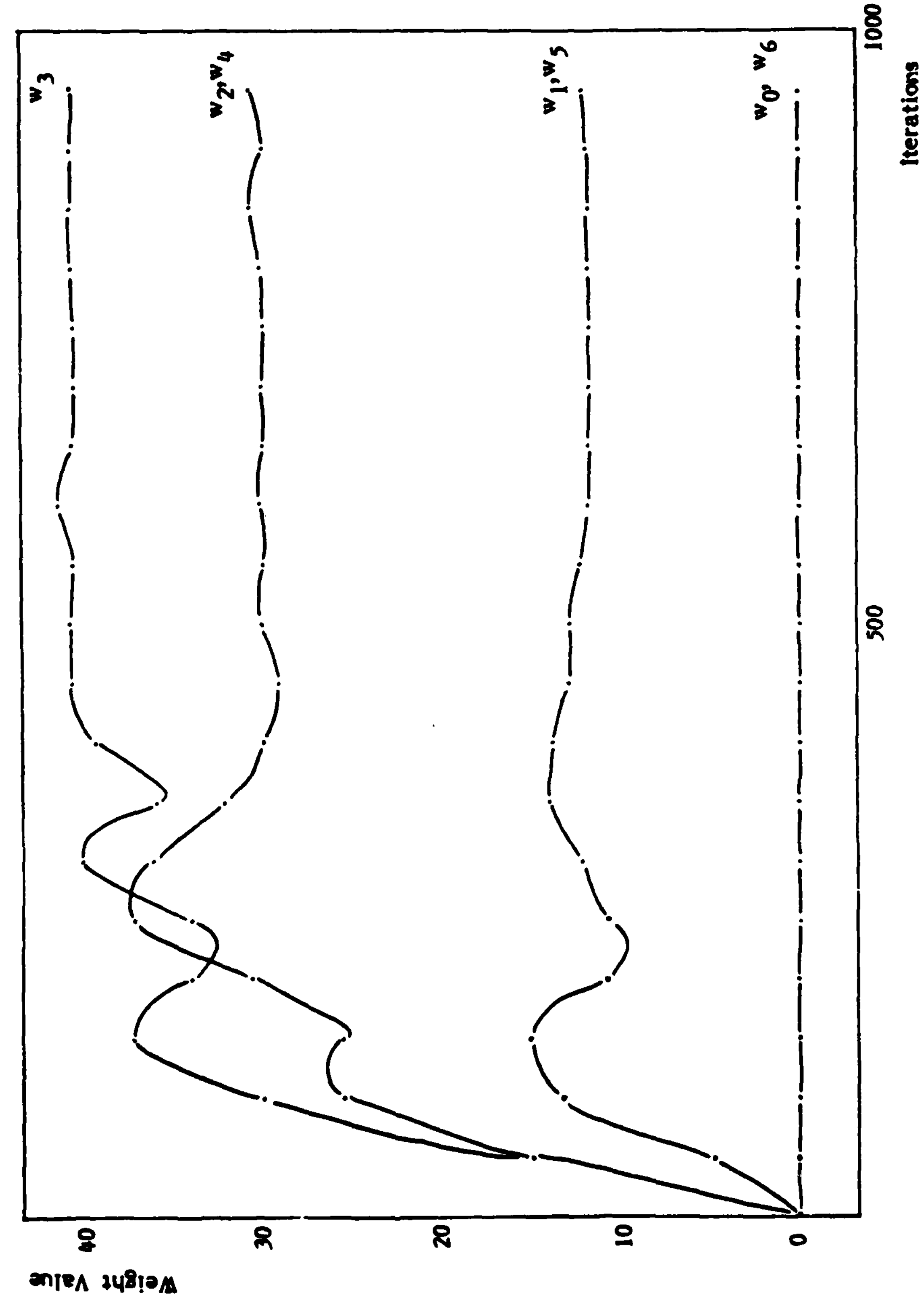
Error and Adaptive Weight Plots

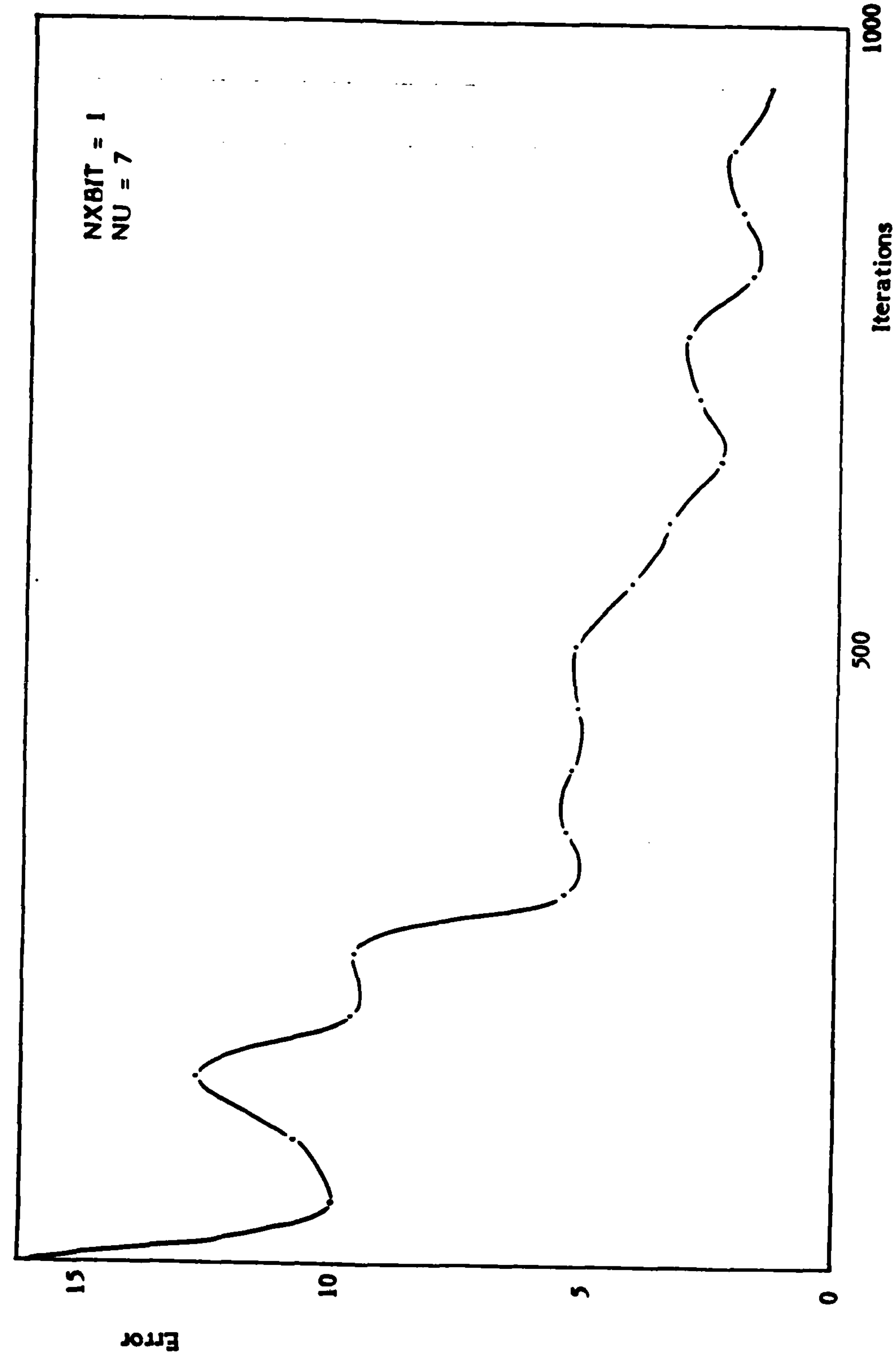
Contained here are the error and adaptive weight curves for each shown in Table 2. These were obtained from simulations of each combination of NXBIT and NU run at their optimum step size. The filter weights should converge to normalized, scaled versions of those in section 2.3 of Chapter II. These values are given in Table 3.

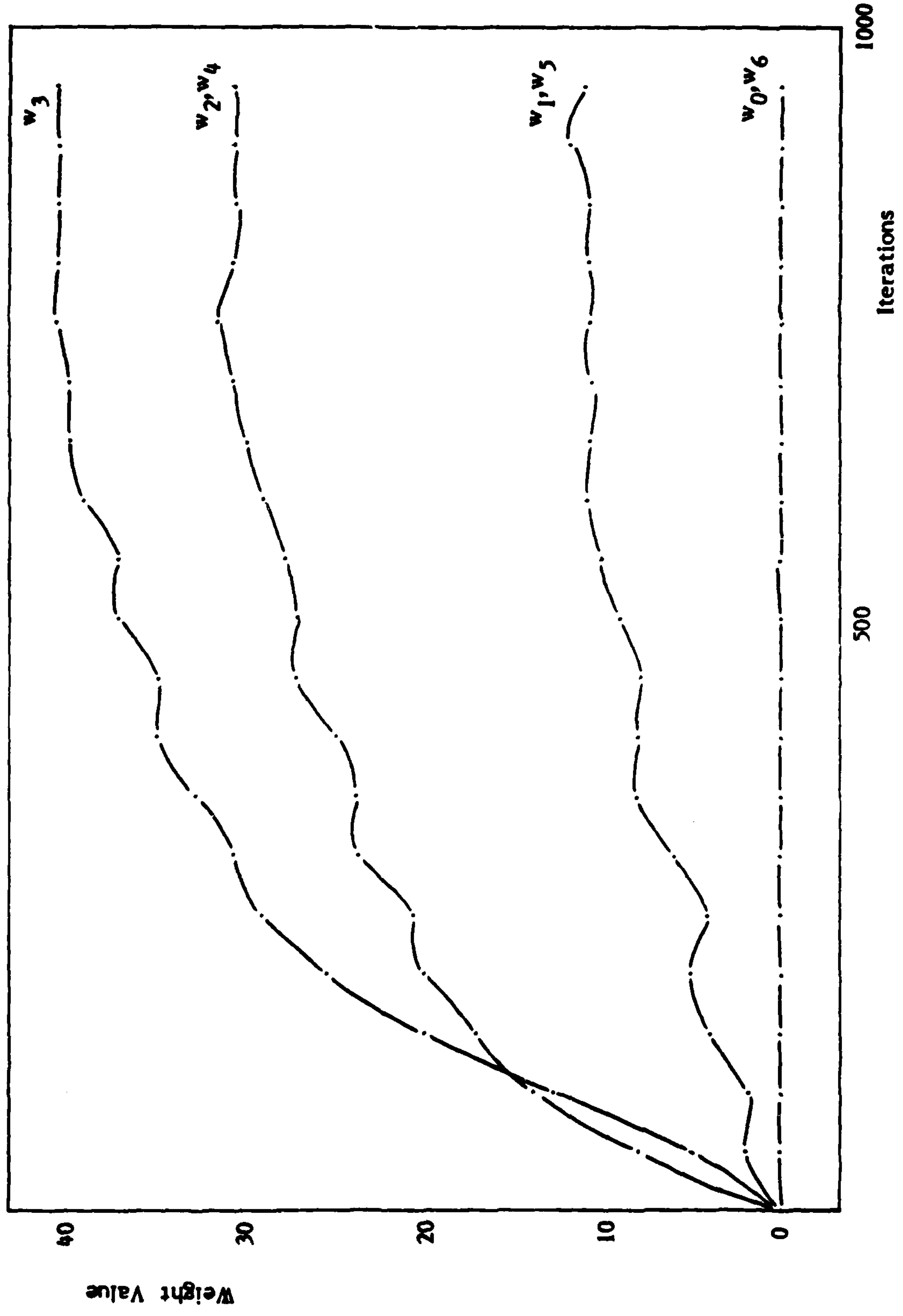
Table 3.
Normalized, Scaled Weights

NU	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
7	2	12	30	40	30	12	2		
8	1	7	21	35	35	21	7	1	
9	1	4	14	28	35	28	14	4	1

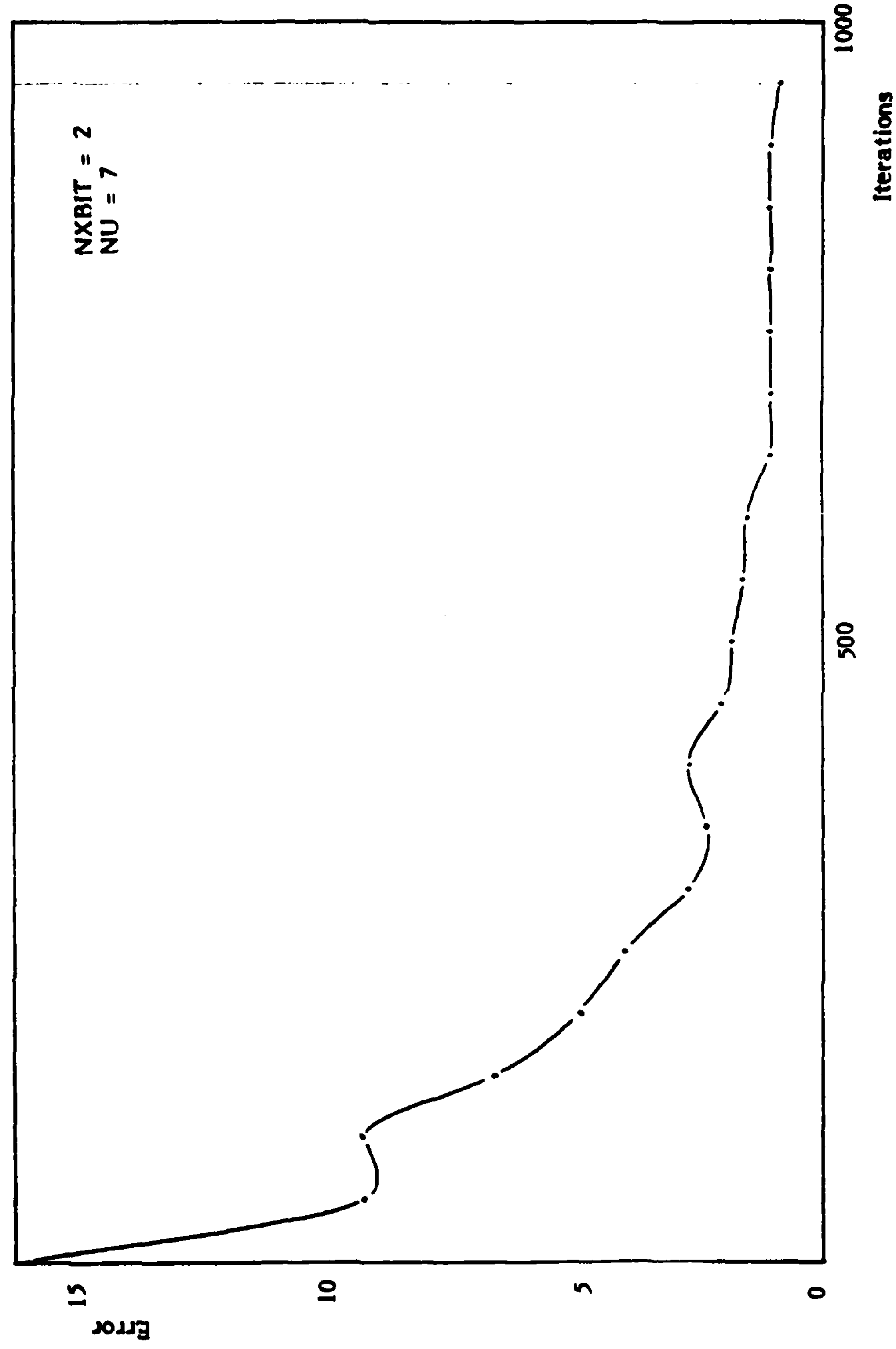

NXBIT = 0
NU = 7
Error
Iterations
15
10
5
0
500
1000


Weight Value
40
30
20
10
0
500
1000
Iterations
w_3
w_2, w_4
w_1, w_5
w_0, w_6

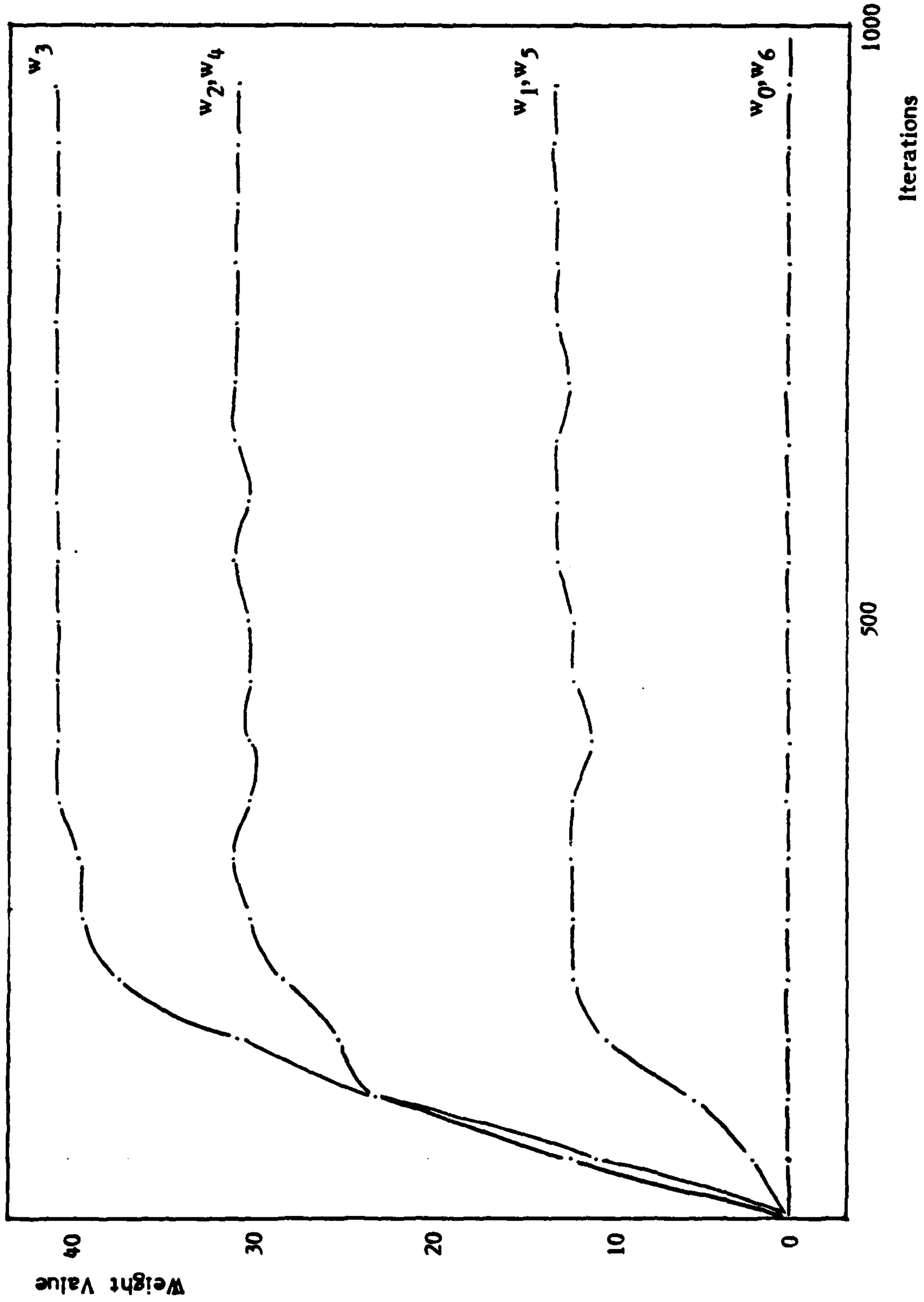

NXBIT = 1
NU = 7
Error
15
10
5
0
500
1000
Iterations

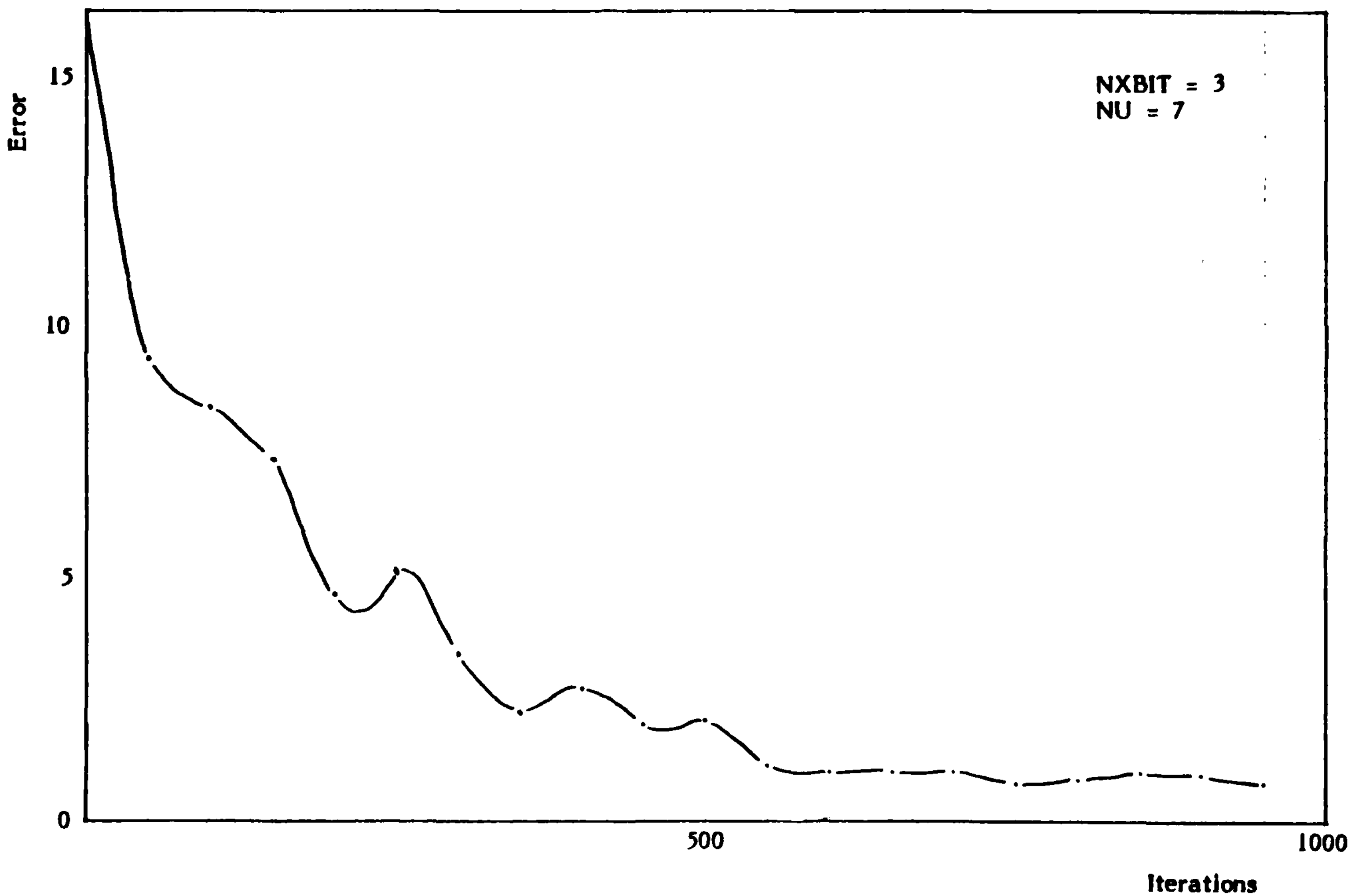


Weight Value
0
10
20
30
40
Iterations
500
1000
w3
w2,w4
w1,w5
w0,w6

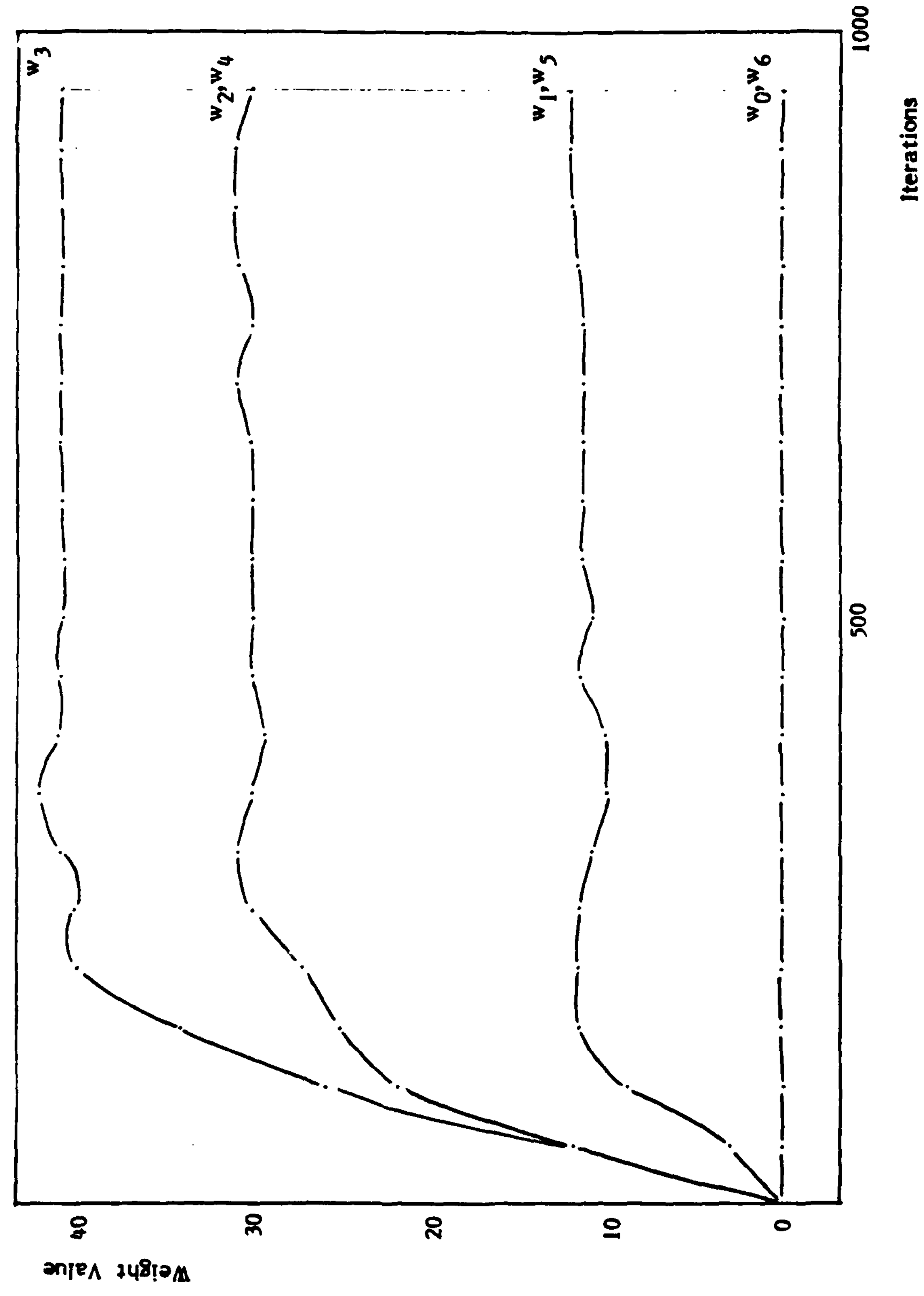


NXBIT = 2
NU = 7
Error
15
10
5
0
Iterations
500
1000

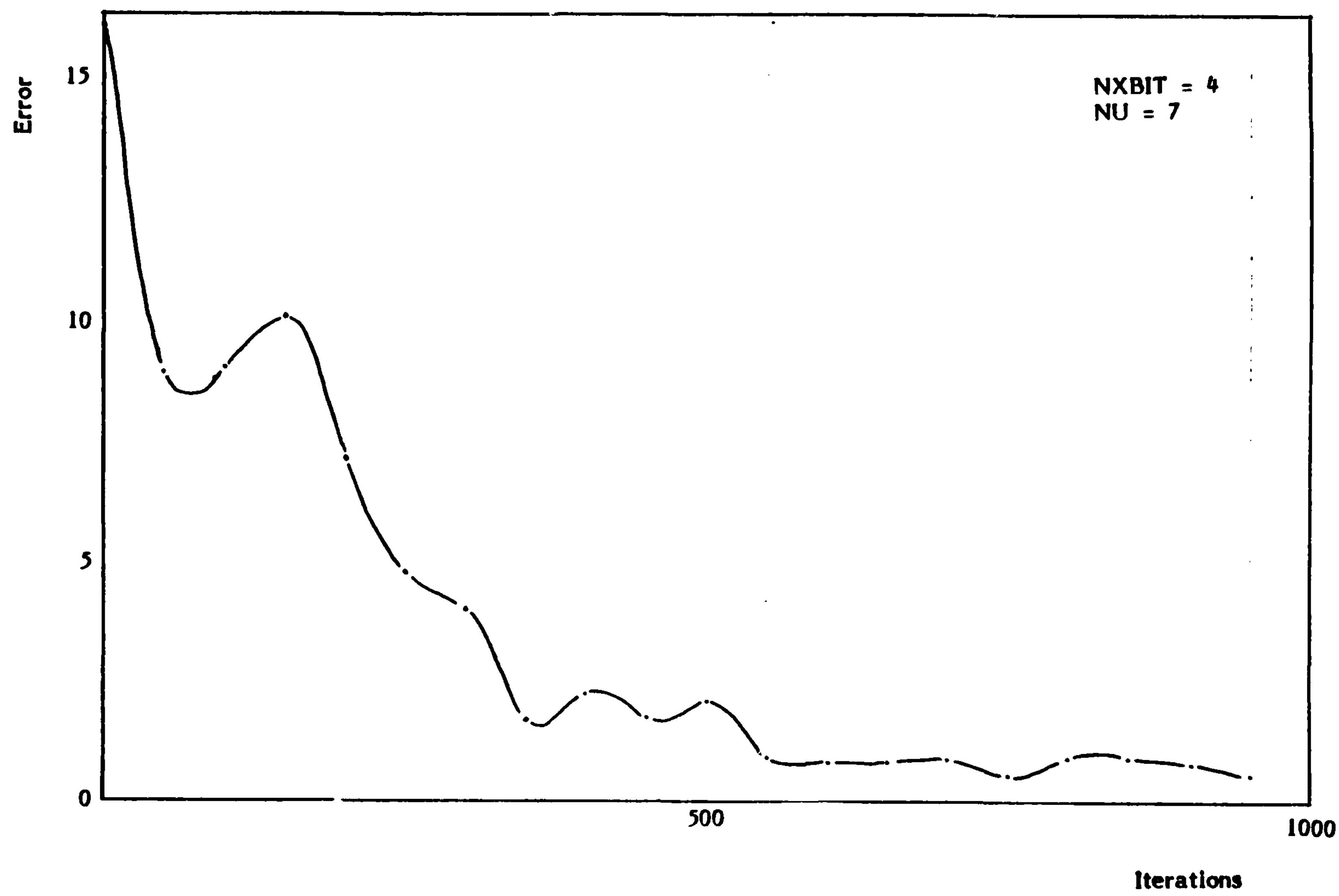

Weight Value
0
10
20
30
40
Iterations
500
1000
w_3
w_2, w_4
w_1, w_5
w_0, w_6

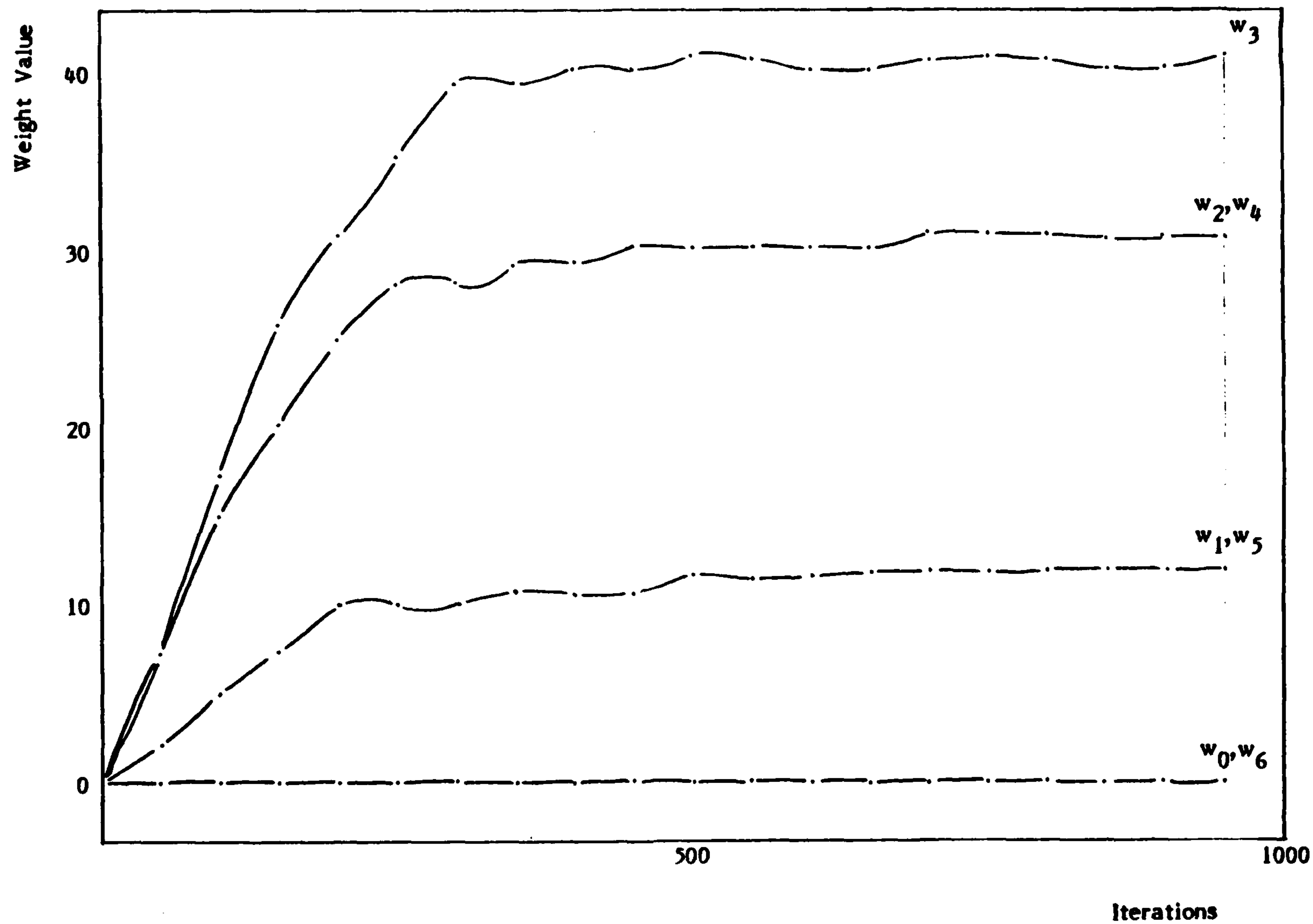


Error
15
10
5
0
500
1000
Iterations
NXBIT = 3
NU = 7

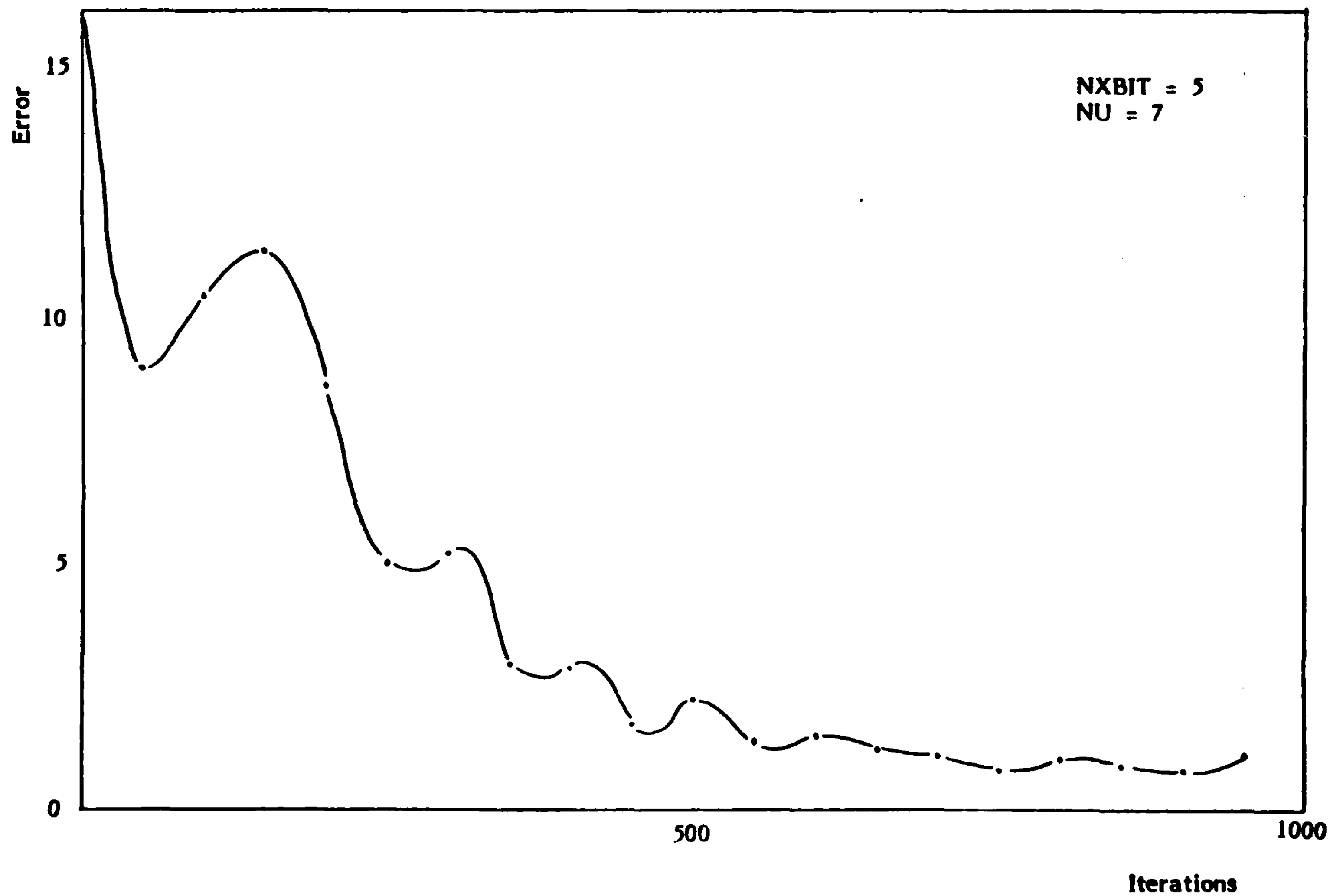


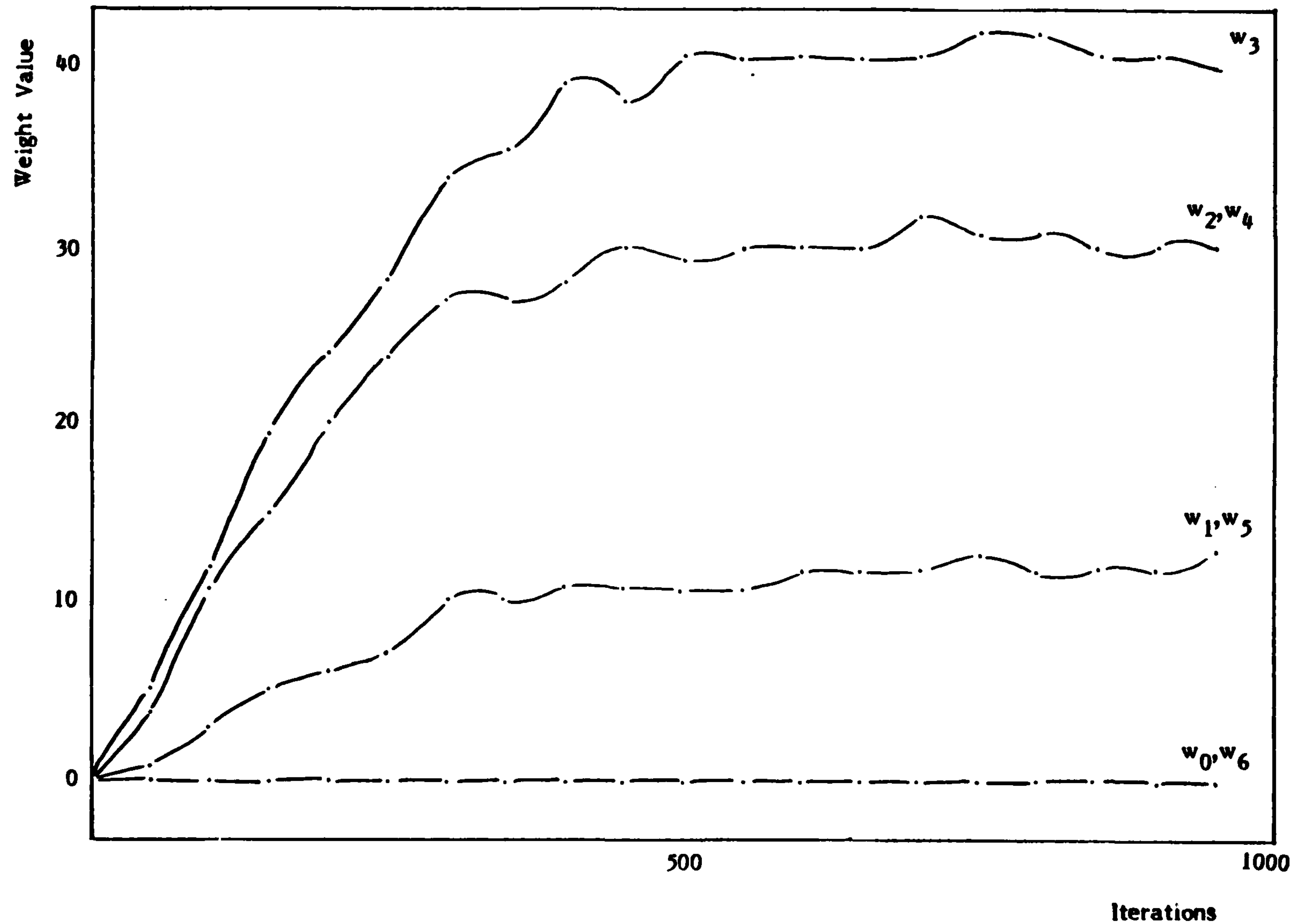
Weight Value
40
30
20
10
0
Iterations
500
1000
w3
w2, w4
w1, w5
w0, w6


Error
15
10
5
0
500
1000
Iterations
NXBIT = 4
NU = 7

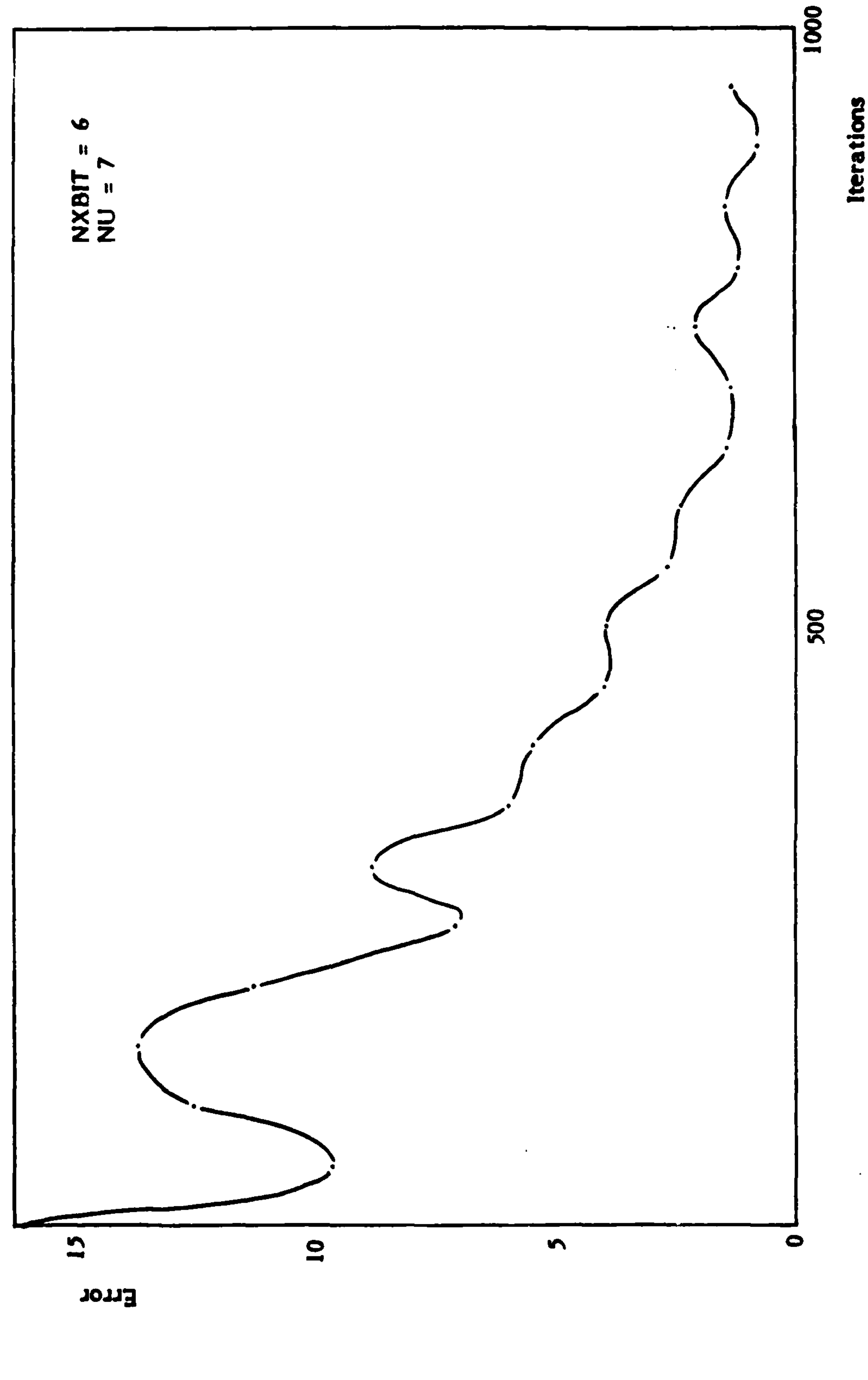


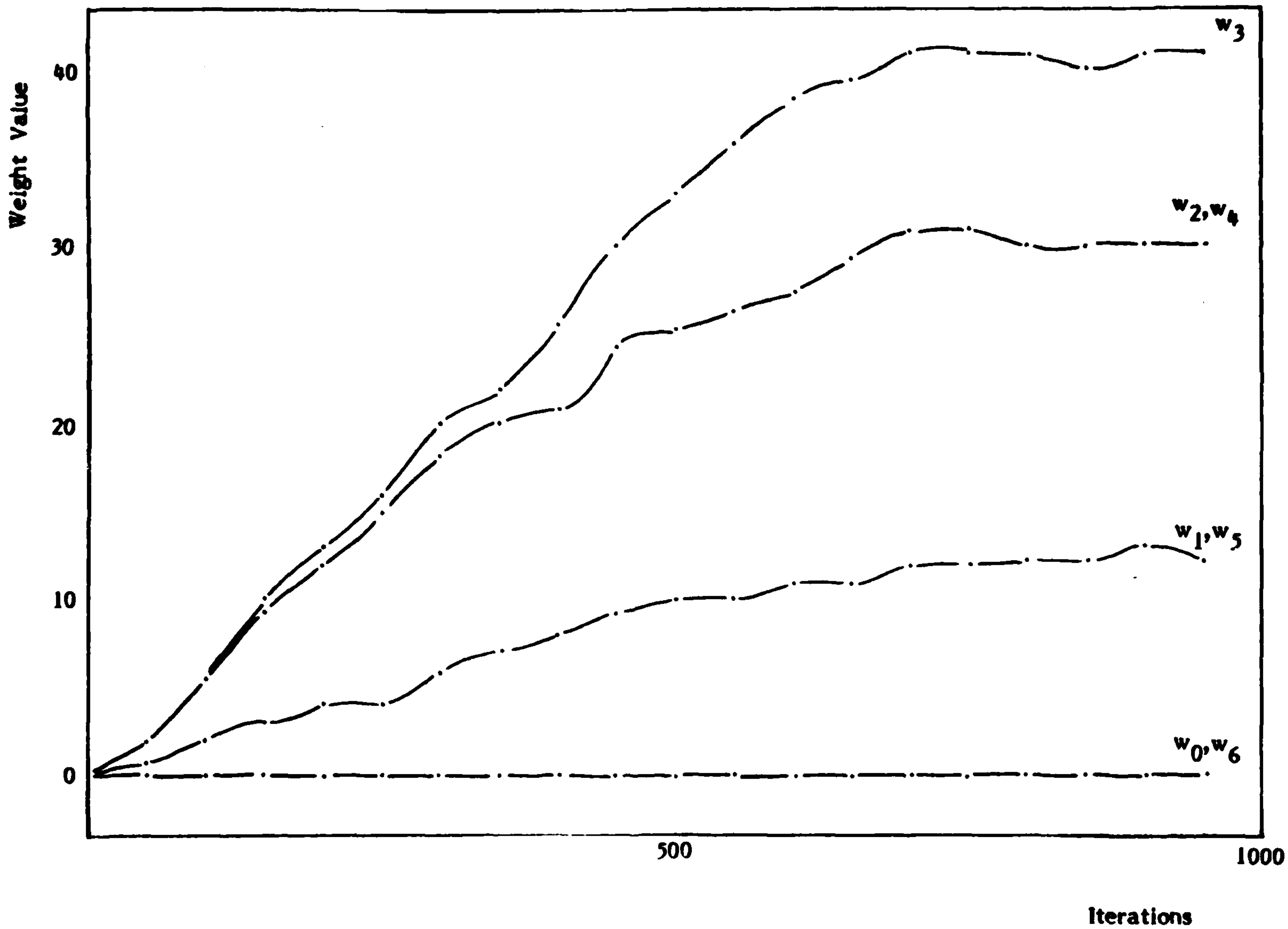
Weight Value
40
30
20
10
0
500
1000
Iterations
w_3
w_2, w_4
w_1, w_5
w_0, w_6

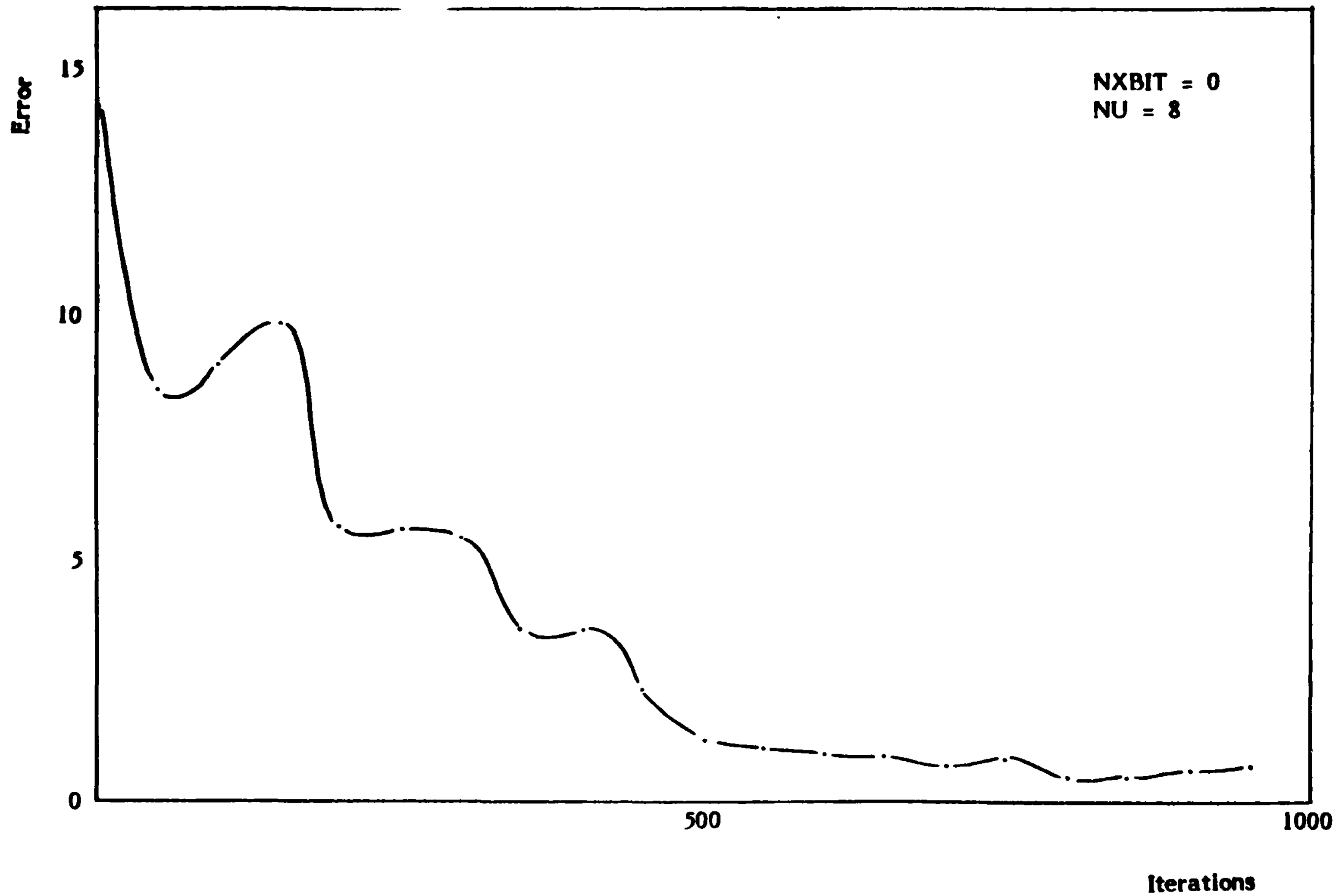

NXBIT = 5
NU = 7
Error
15
10
5
0
500
1000
Iterations



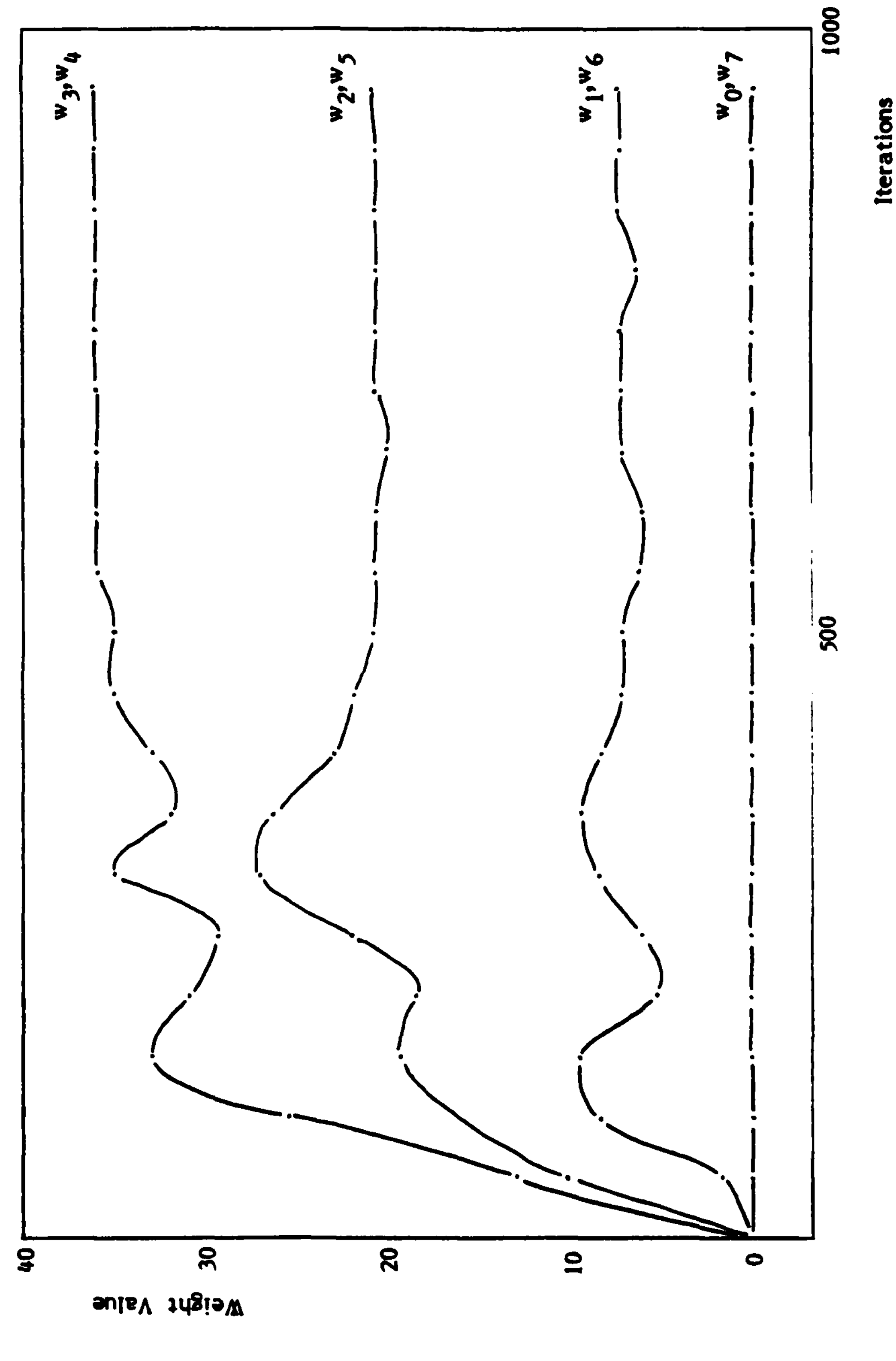
Weight Value
40
30
20
10
0
500
1000
Iterations
w3
w2,w4
w1,w5
w0,w6

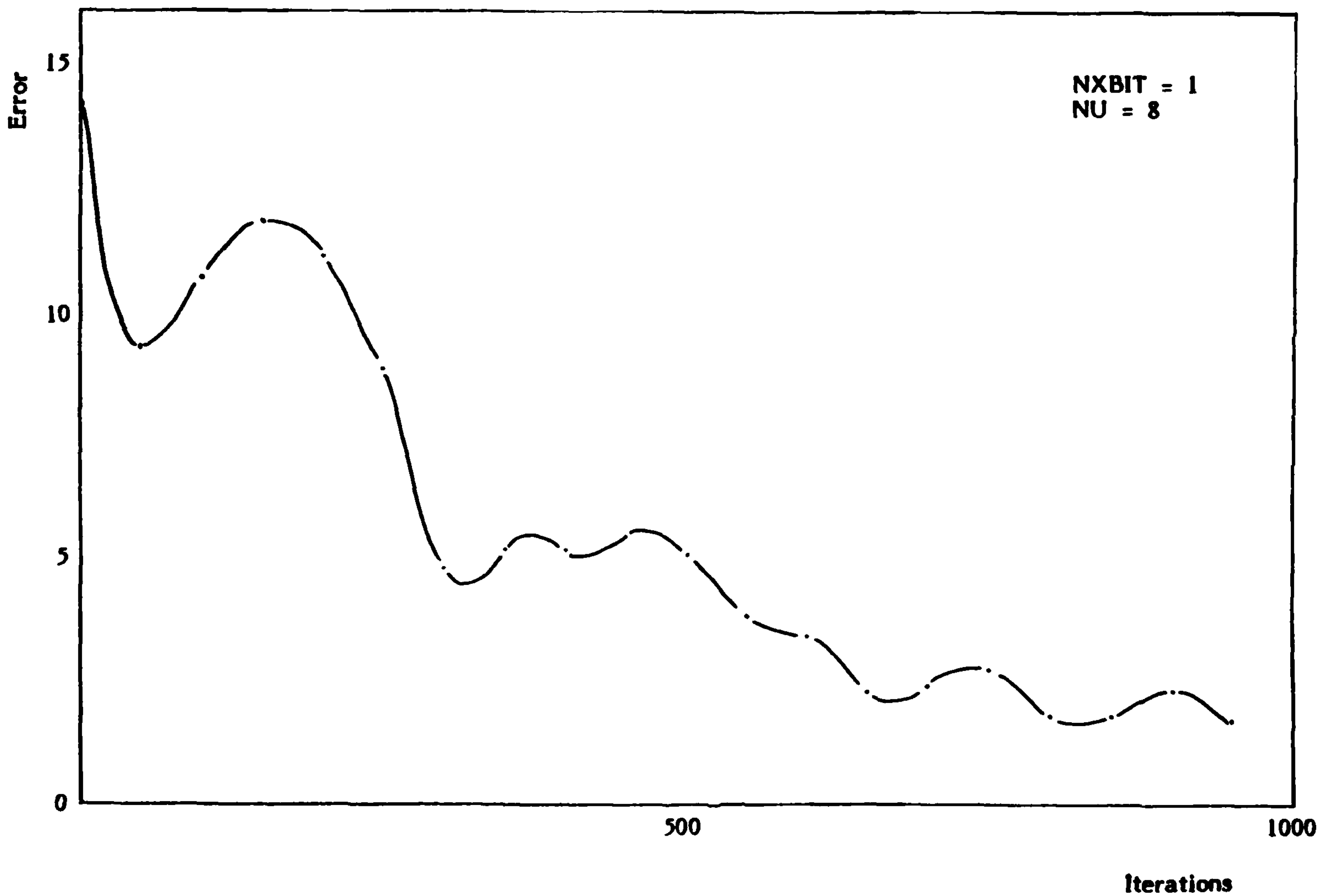

NXBIT = 6
NU = 7
Error
Iterations
0
5
10
15
500
1000

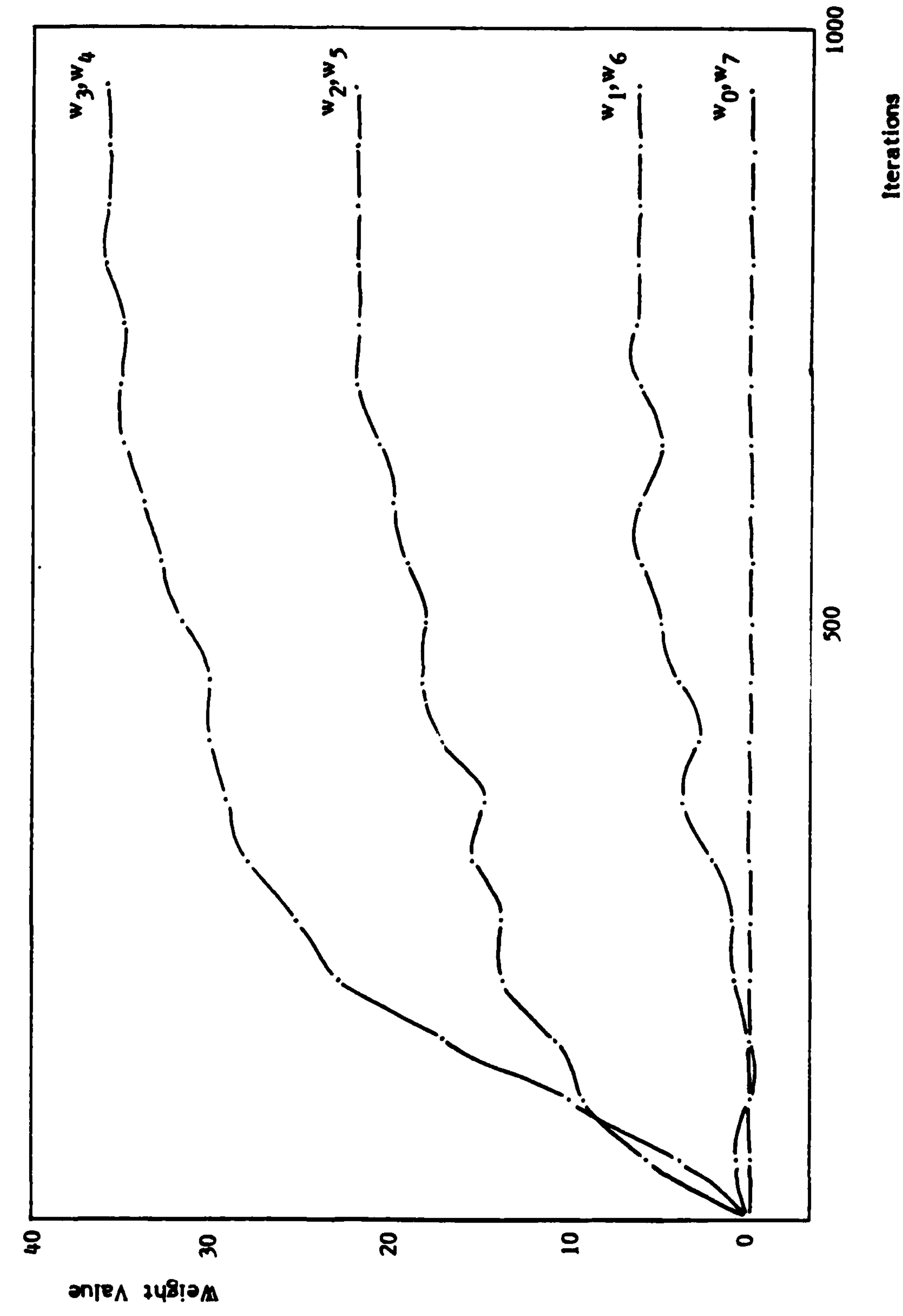

Weight Value
40
30
20
10
0
500
1000
Iterations
w_3
w_2, w_4
w_1, w_5
w_0, w_6

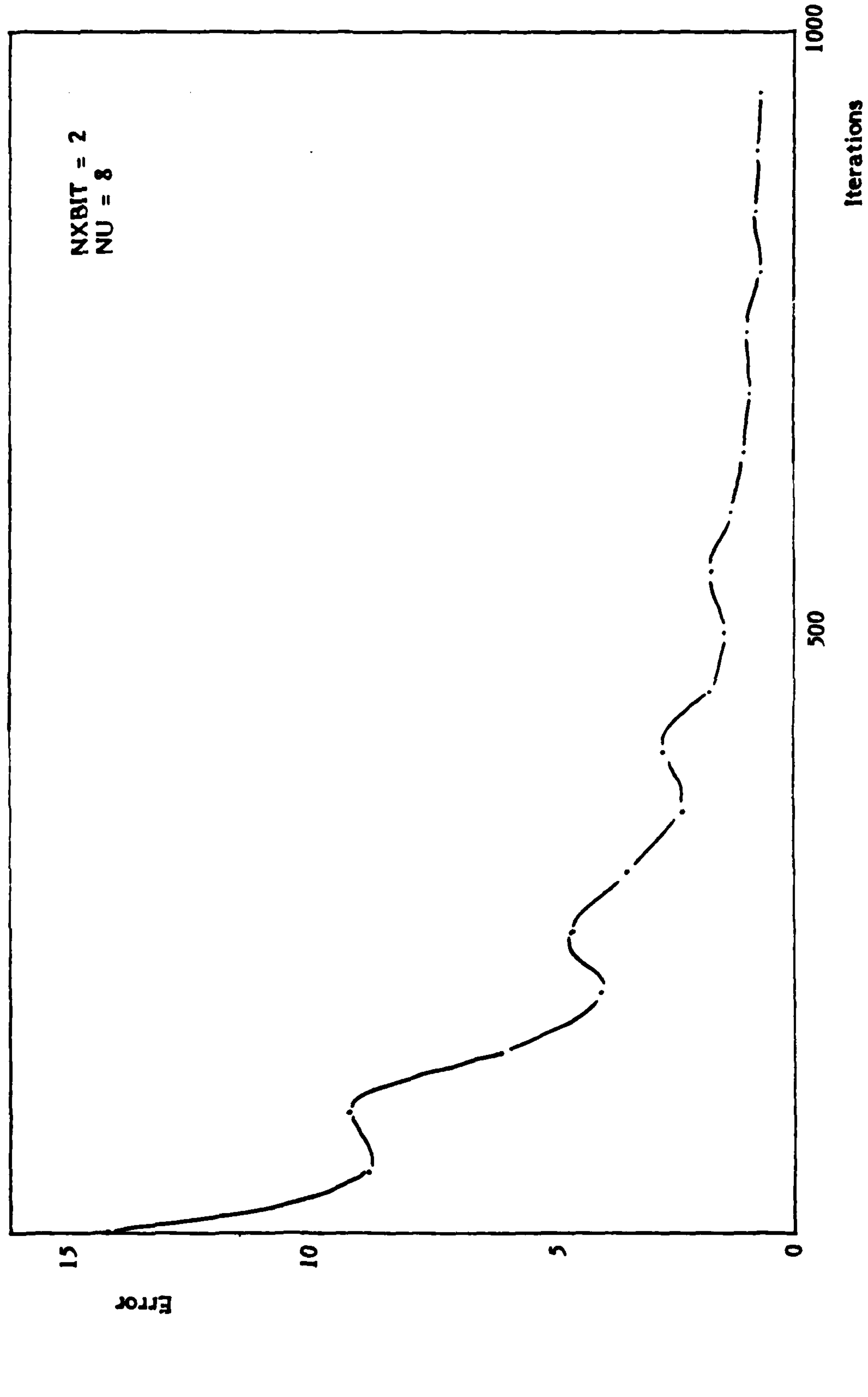


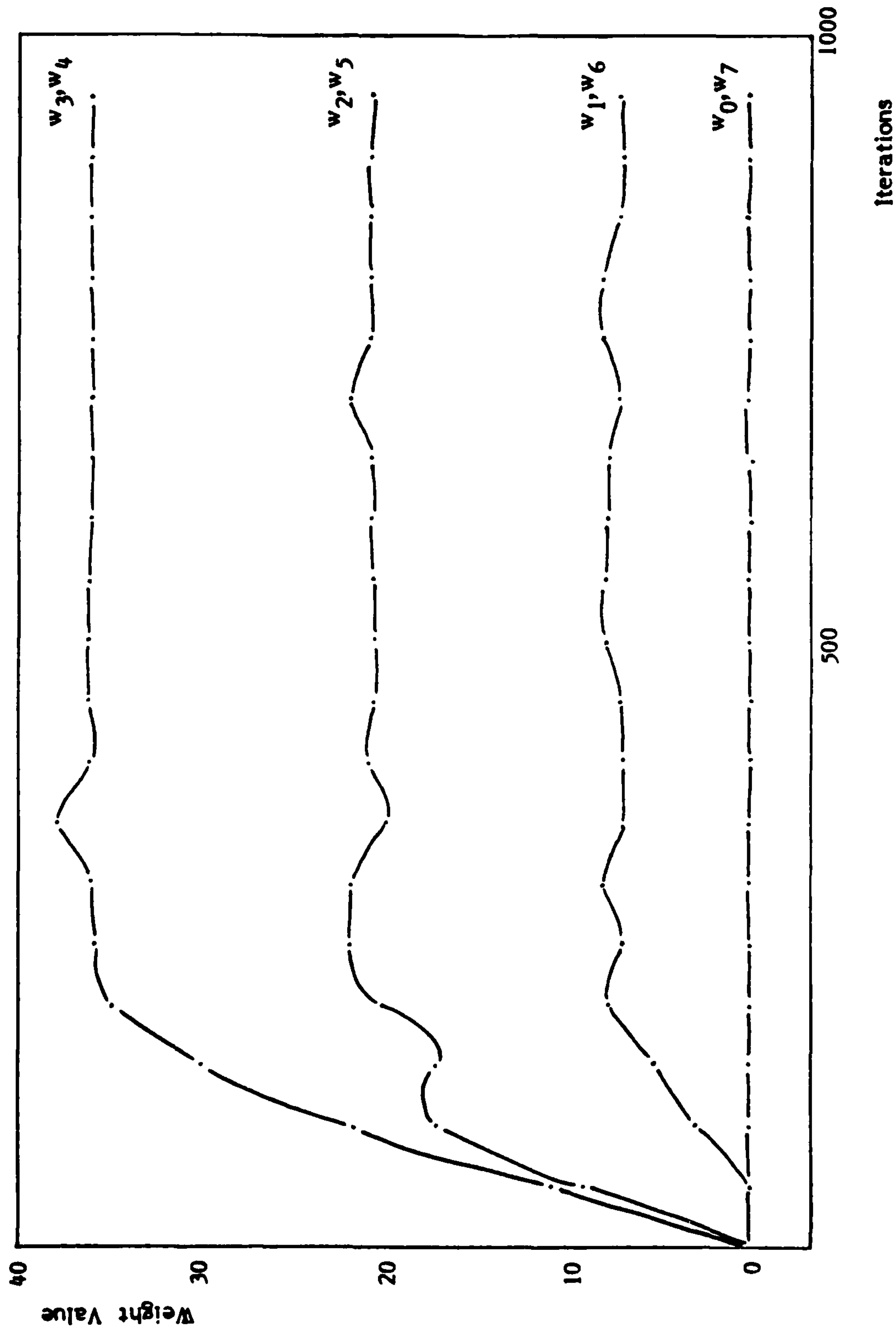
15
10
5
0
Error
500
1000
Iterations
NXBIT = 0
NU = 8


w_3, w_4
w_2, w_5
w_1, w_6
w_0, w_7
0
500
1000
Iterations
0
10
20
30
40
Weight Value

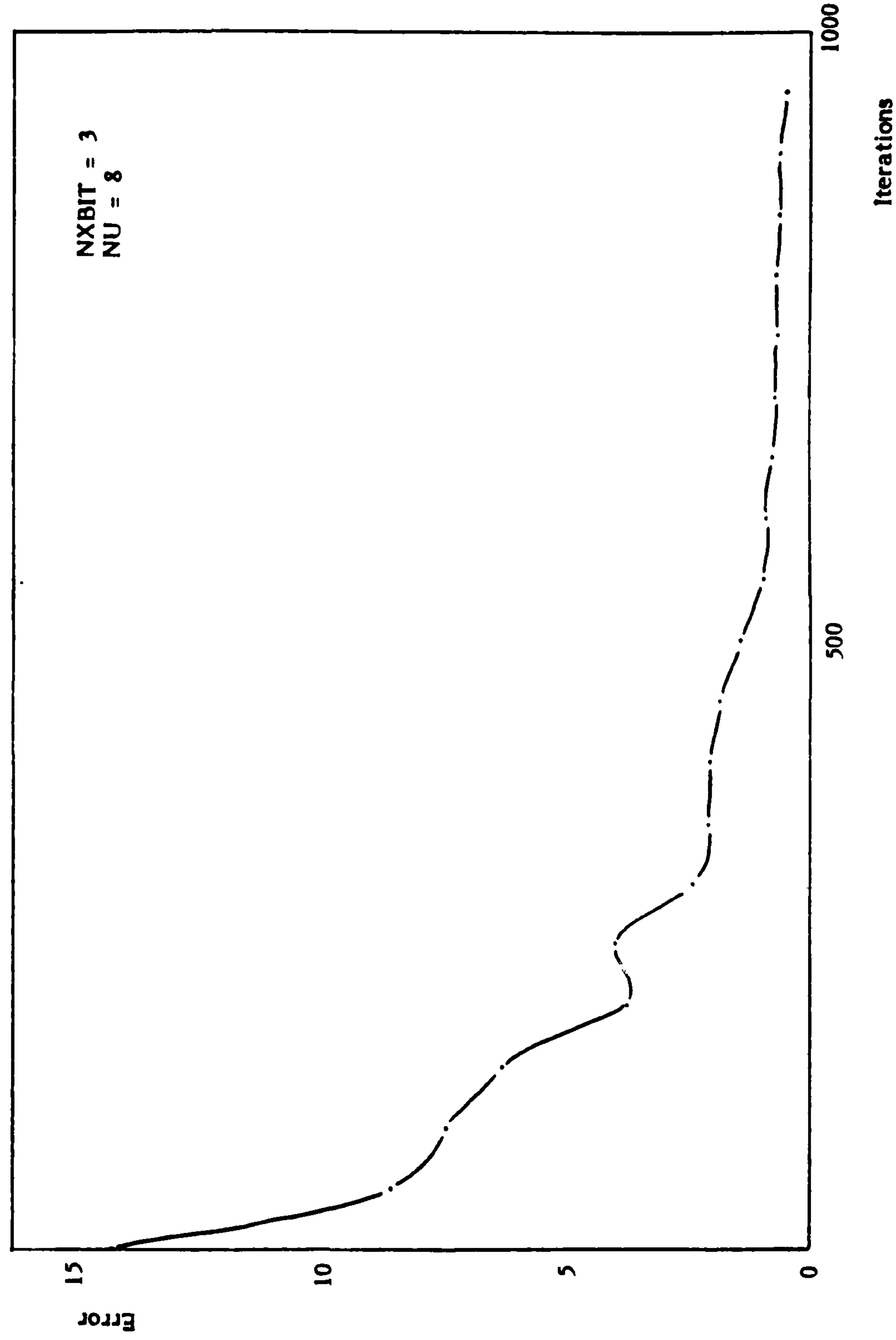

Error
15
10
5
0
500
1000
Iterations
NXBIT = 1
NU = 8

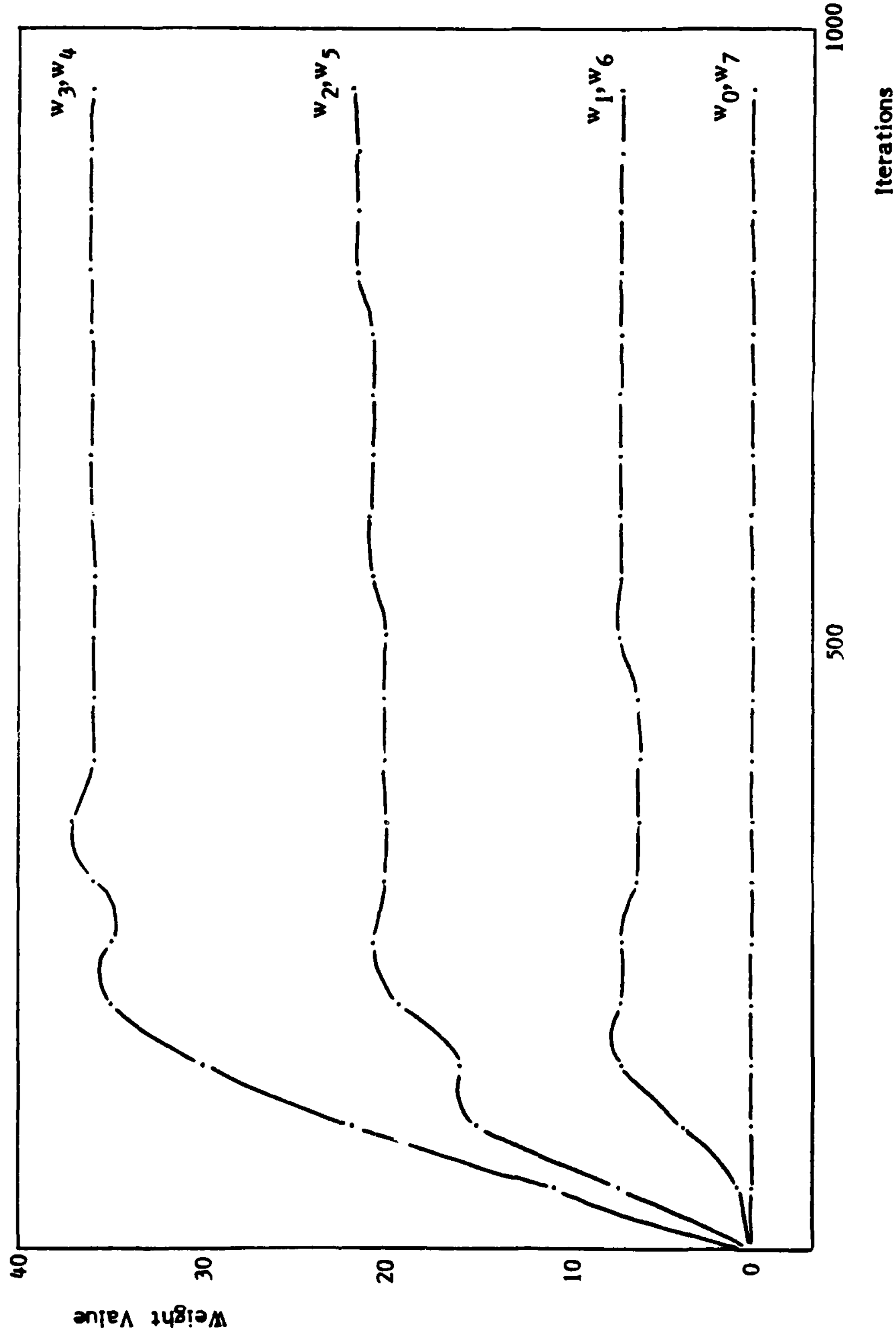

Weight Value
0
10
20
30
40
Iterations
500
1000
w_3, w_4
w_2, w_5
w_1, w_6
w_0, w_7

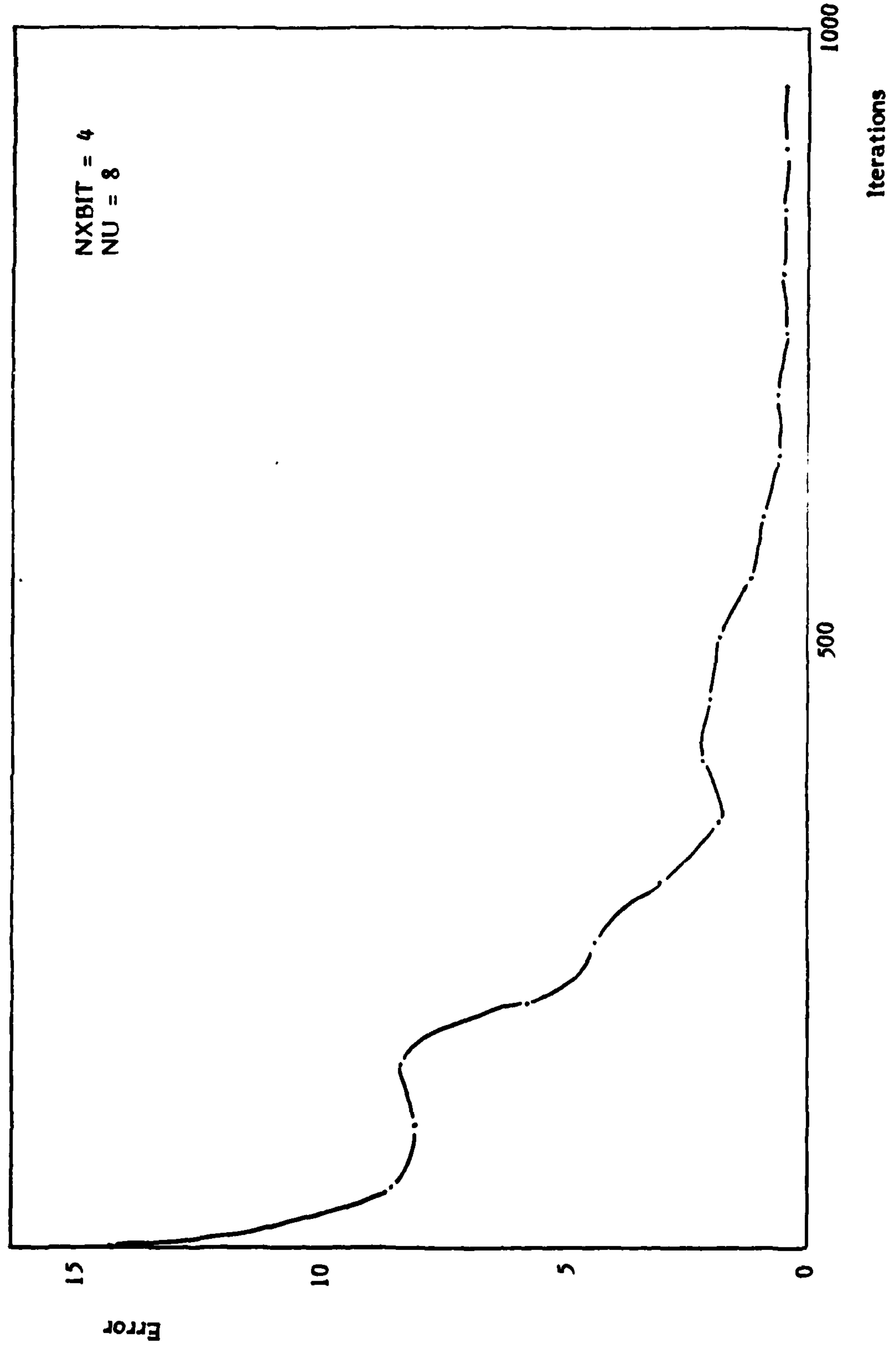

NXBIT = 2
NU = 8
Error
Iterations
0
5
10
15
500
1000

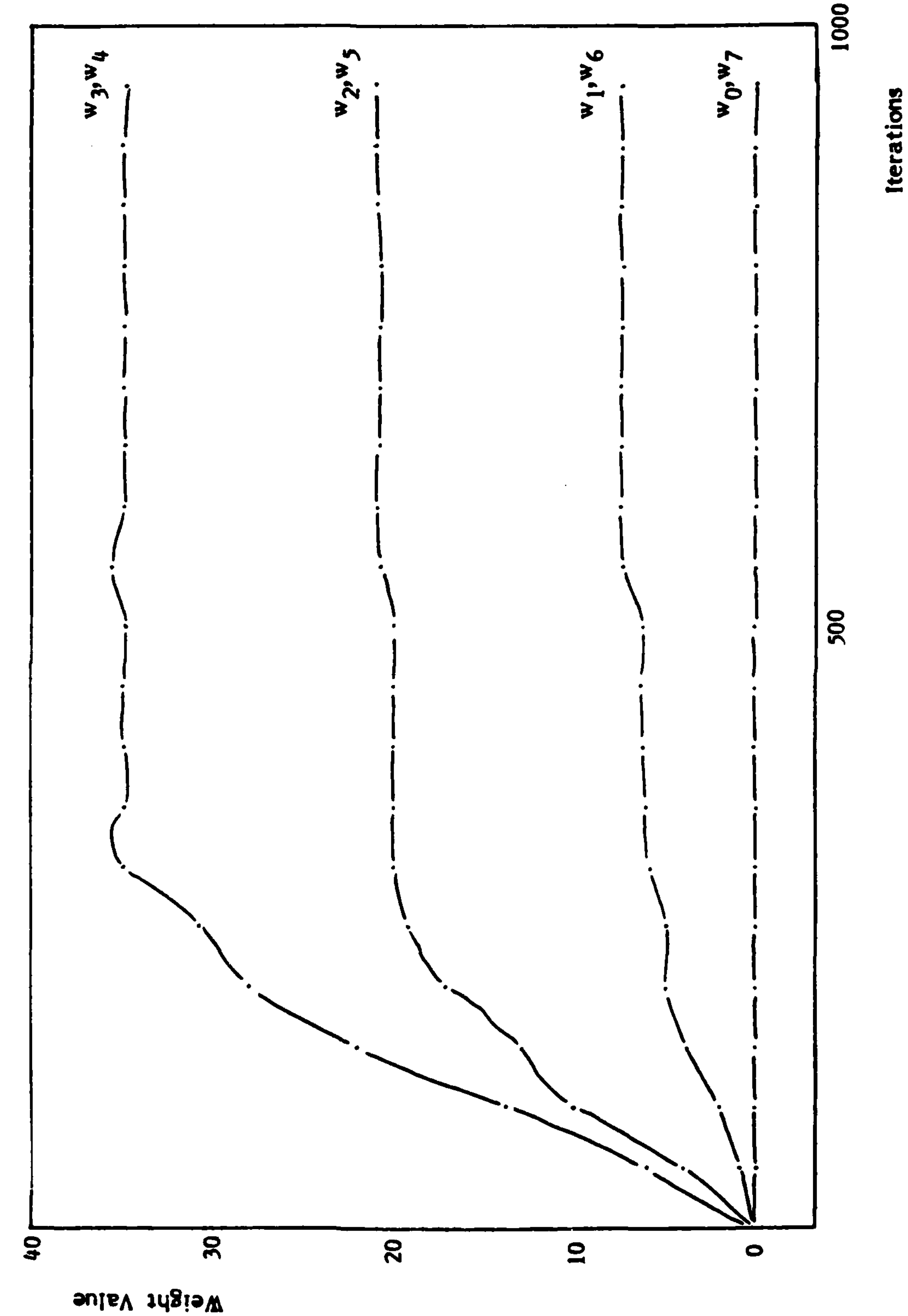


Weight Value
0
10
20
30
40
Iterations
500
1000
w_3, w_4
w_2, w_5
w_1, w_6
w_0, w_7

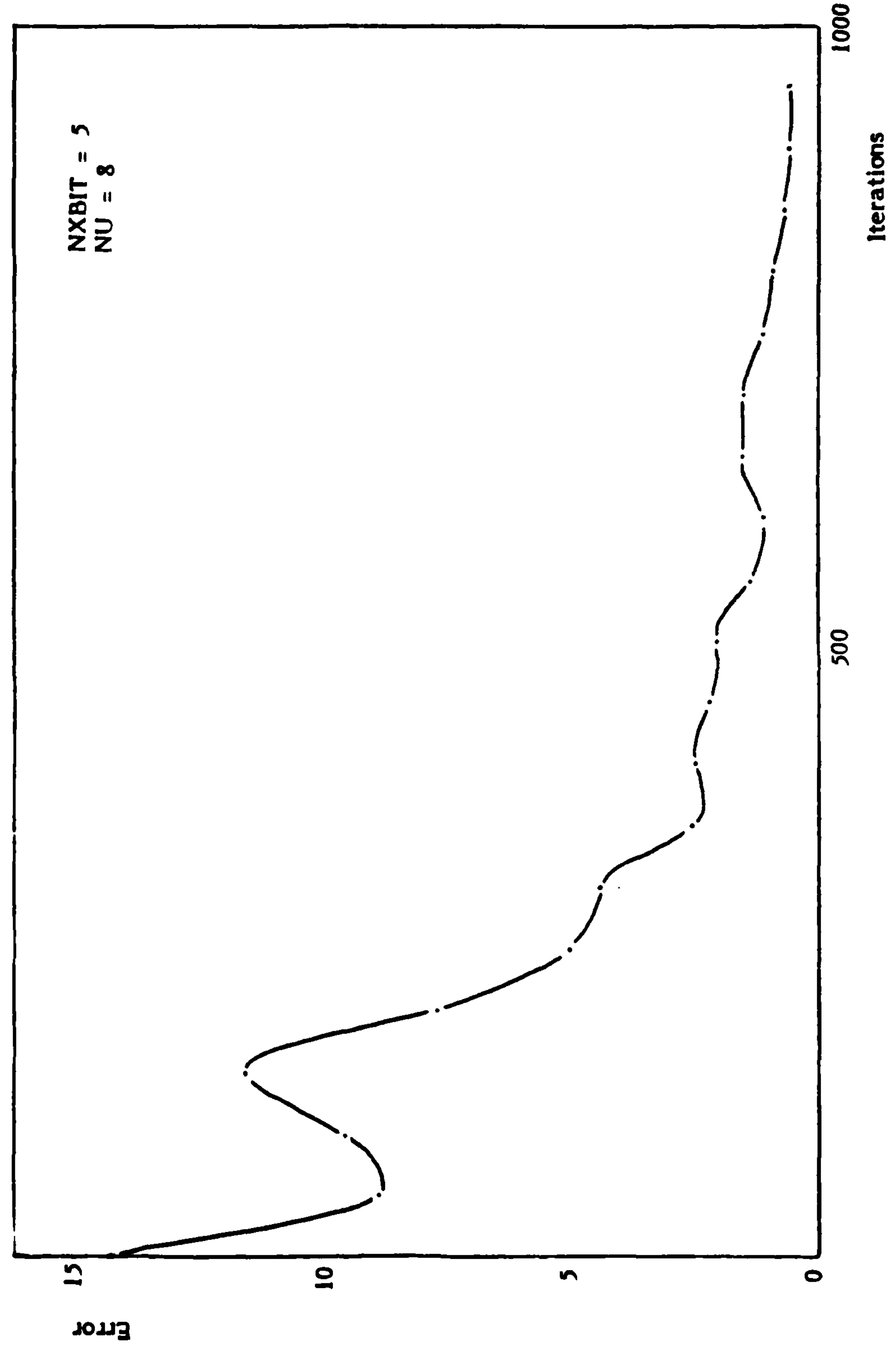

NXBIT = 3
NU = 8
Error
15
10
5
0
500
1000
Iterations


Weight Value
Iterations
0
10
20
30
40
500
1000
w_3, w_4
w_2, w_5
w_1, w_6
w_0, w_7

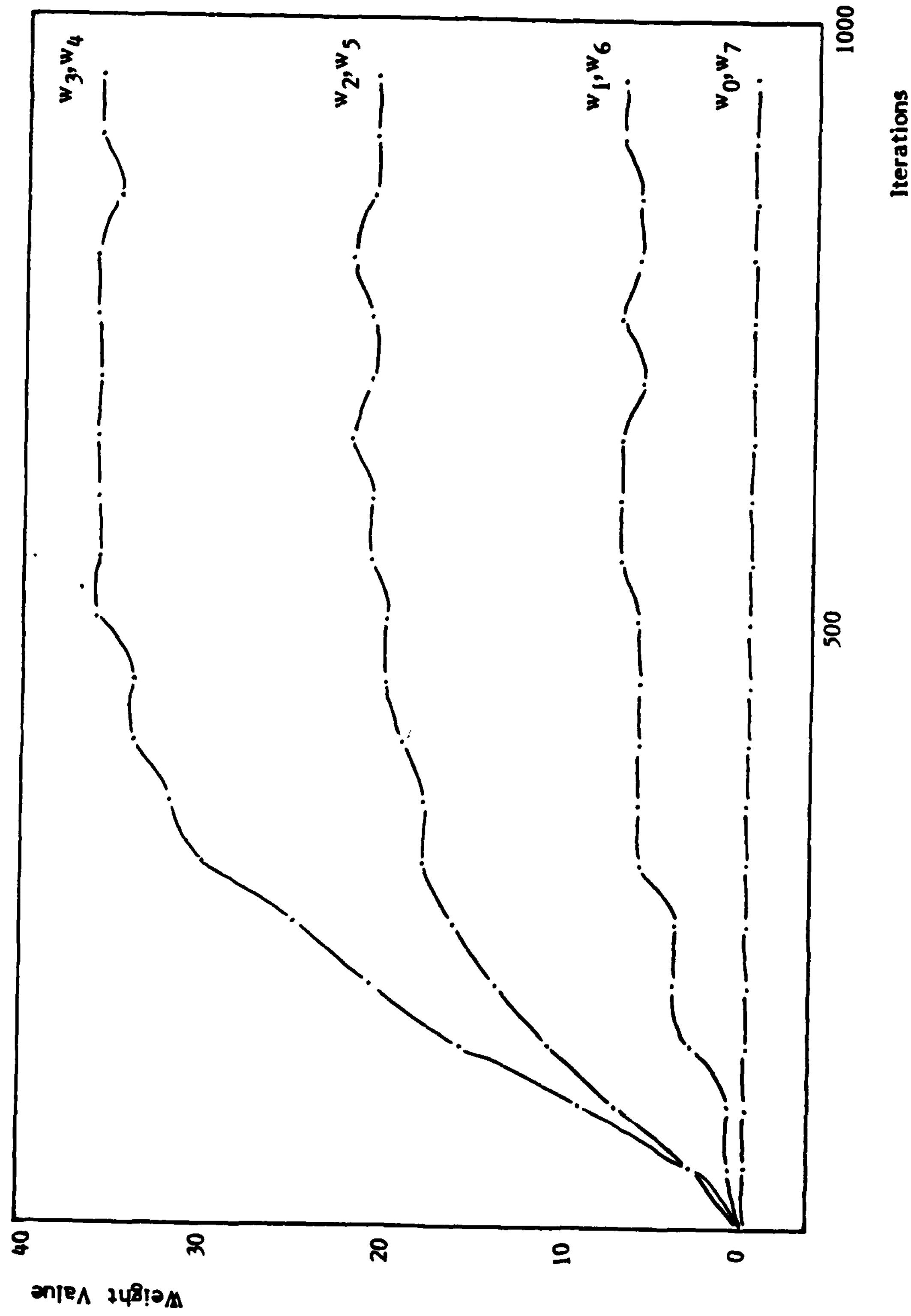

NXBIT = 4
NU = 8
Error
15
10
5
0
Iterations
500
1000

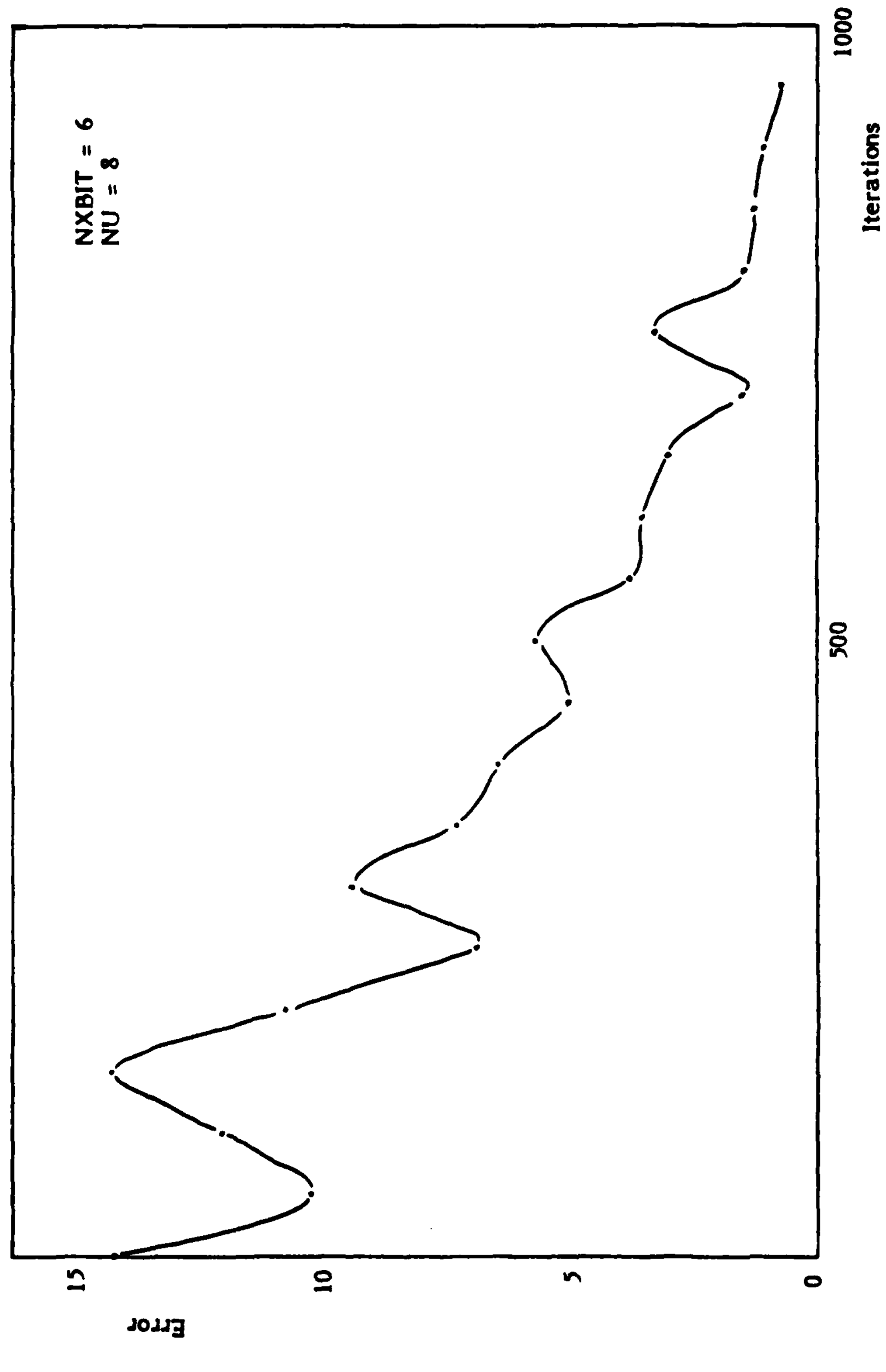


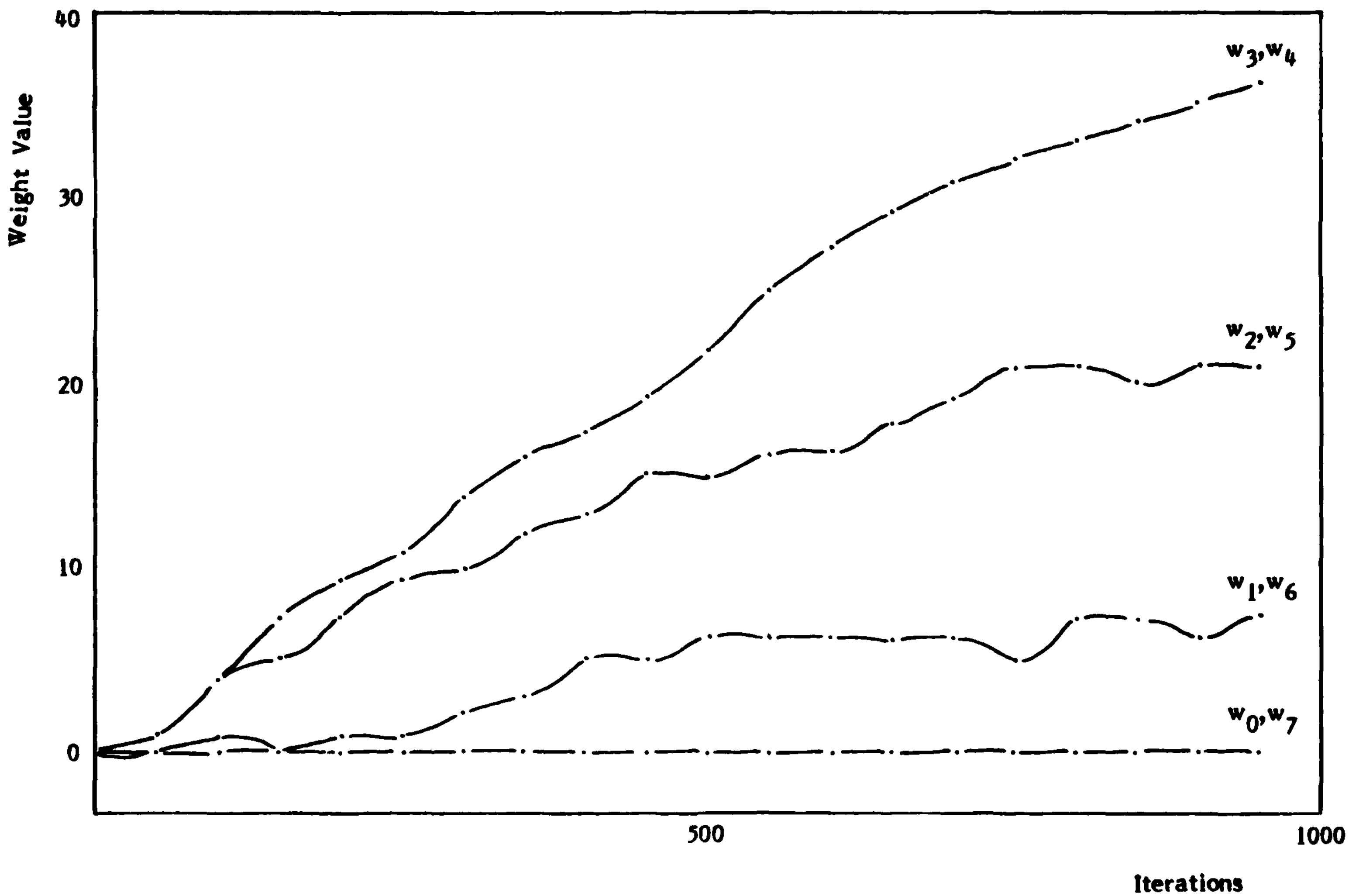
Weight Value
0
10
20
30
40
Iterations
500
1000
w_3, w_4
w_2, w_5
w_1, w_6
w_0, w_7

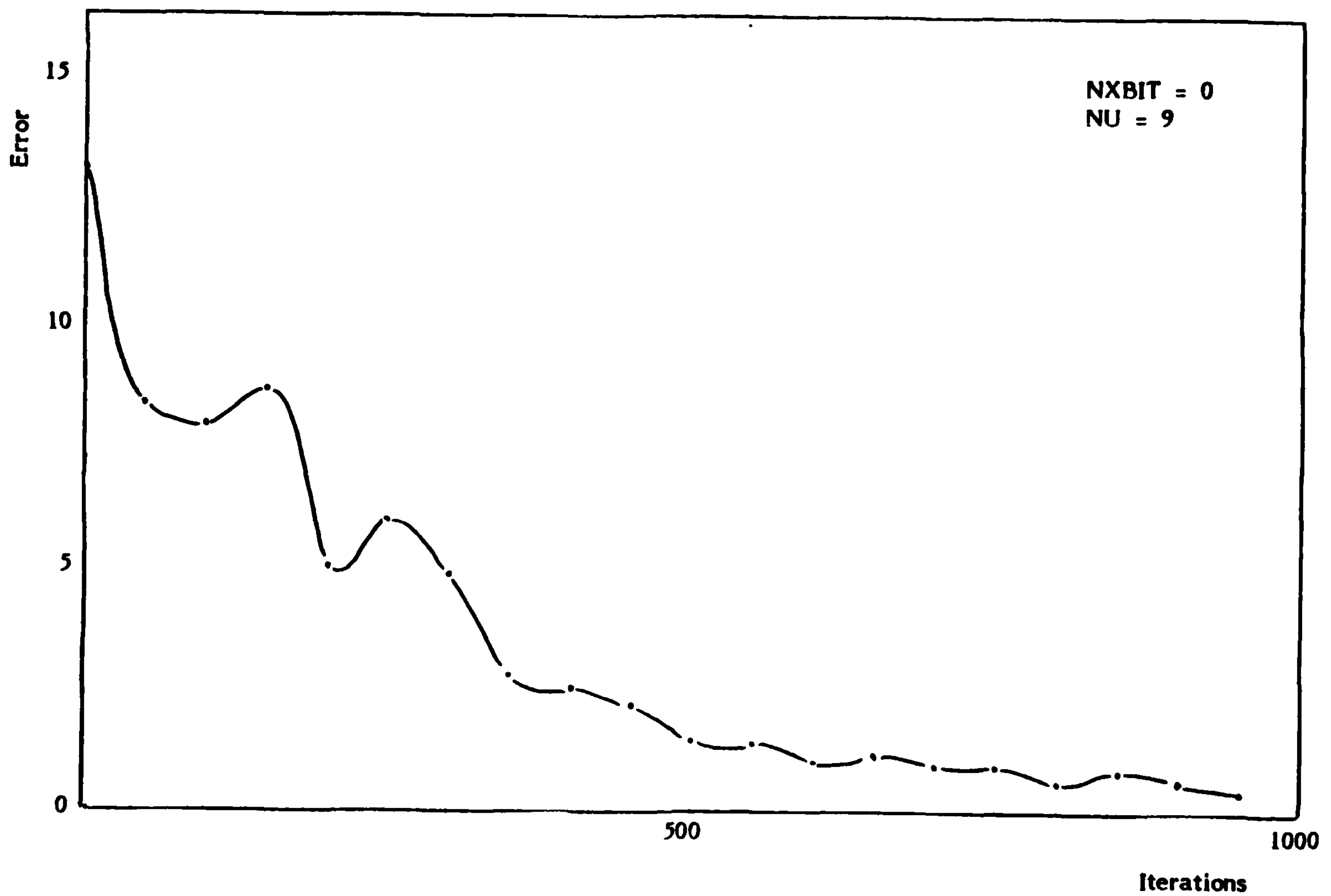


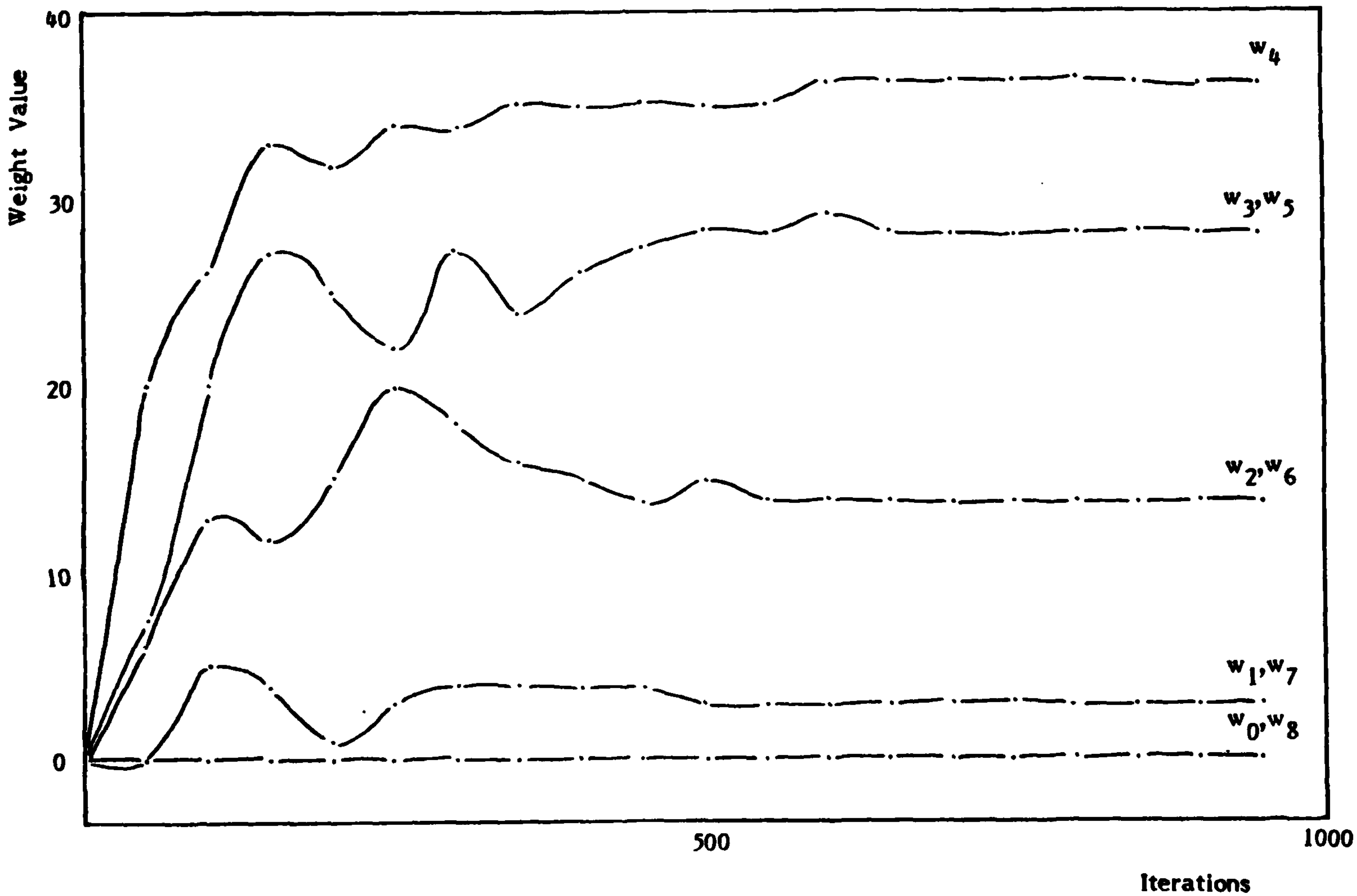
NXBIT = 5
NU = 8
Error
15
10
5
0
500
1000
Iterations


Weight Value
0
10
20
30
40
Iterations
500
1000
w_3, w_4
w_2, w_5
w_1, w_6
w_0, w_7

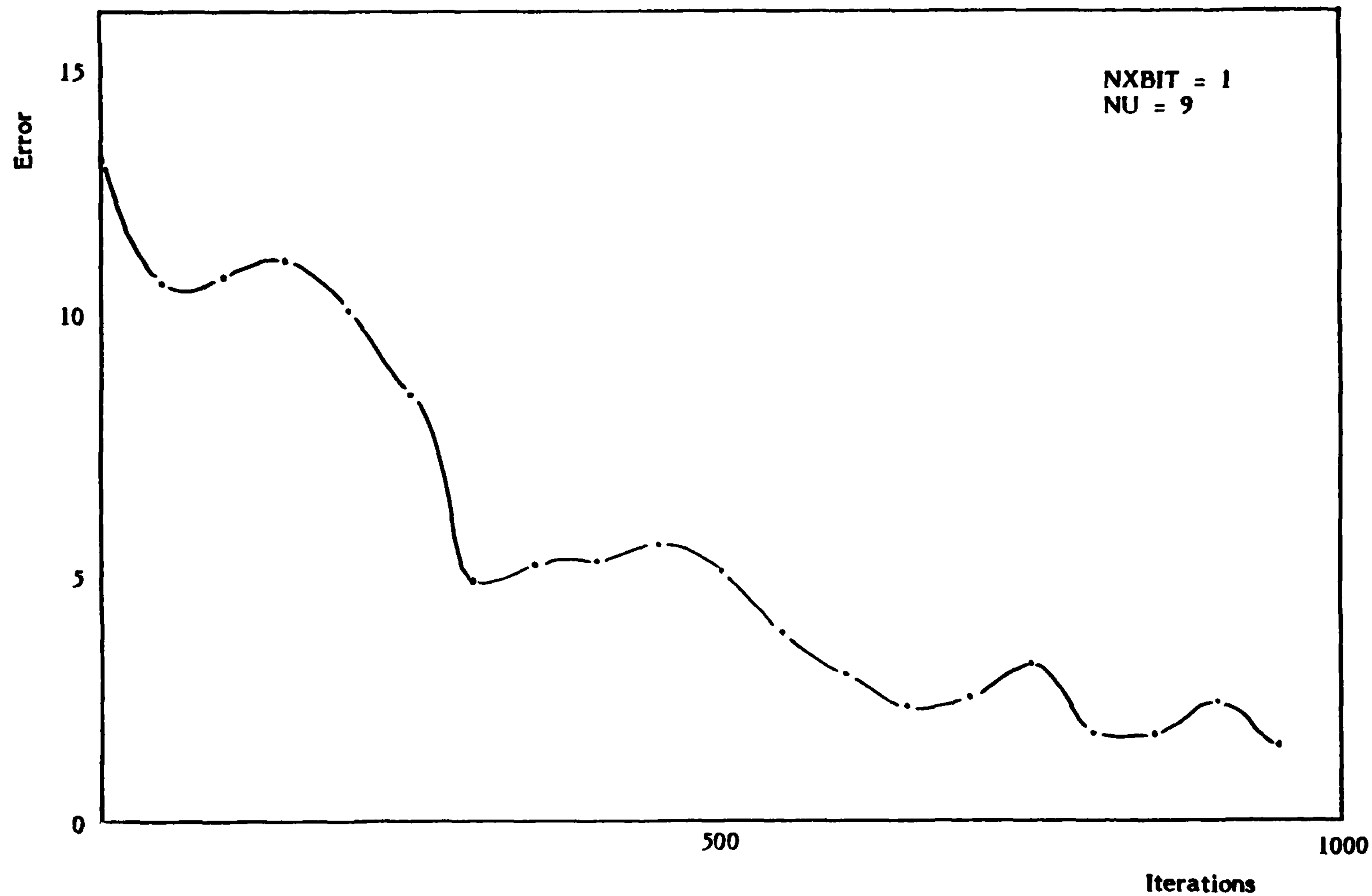

NXBIT = 6
NU = 8
Error
15
10
5
0
500
1000
Iterations

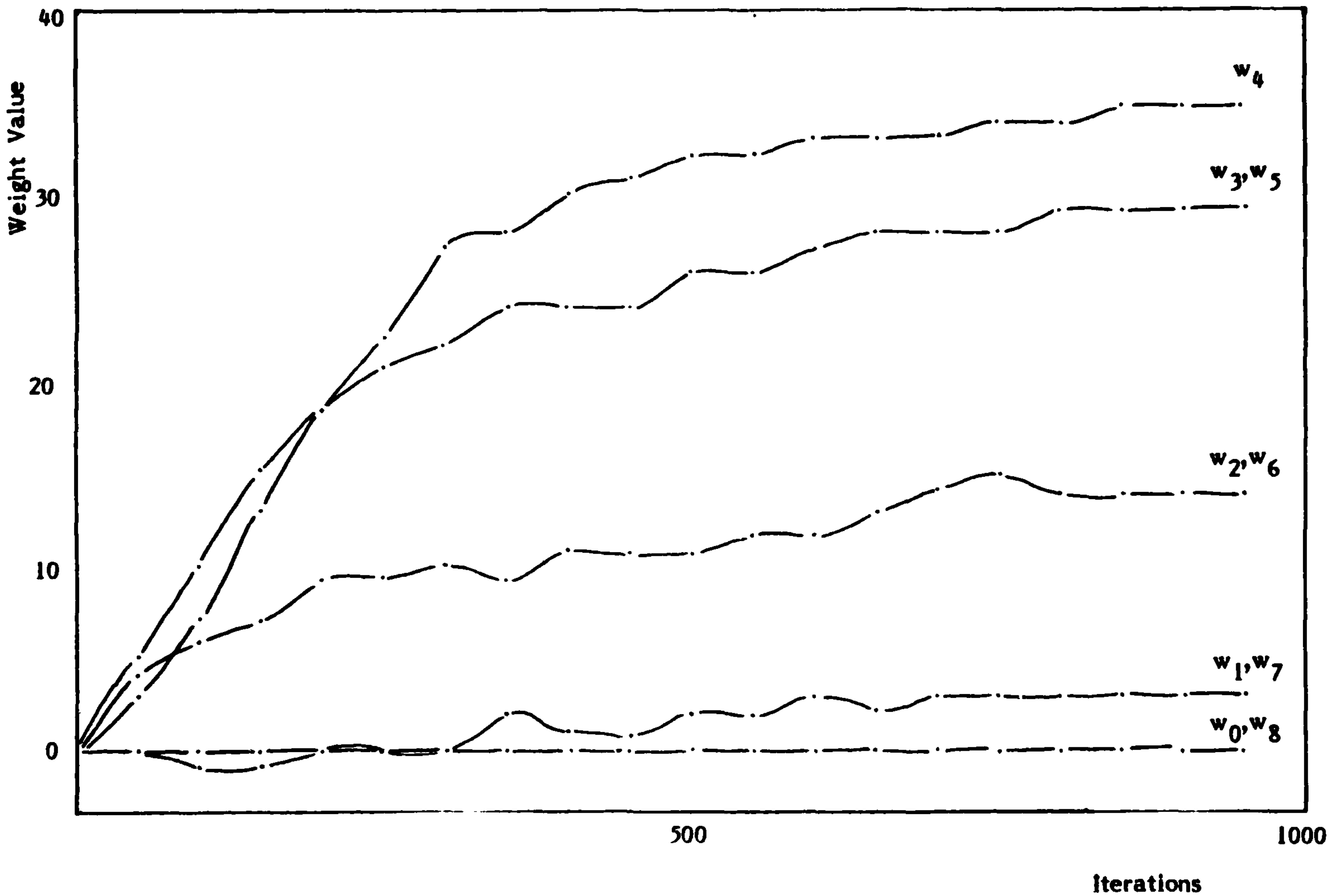

40
30
20
10
0
Weight Value
500
1000
Iterations
w_3, w_4
w_2, w_5
w_1, w_6
w_0, w_7


NXBIT = 0
NU = 9
Error
15
10
5
0
500
1000
Iterations

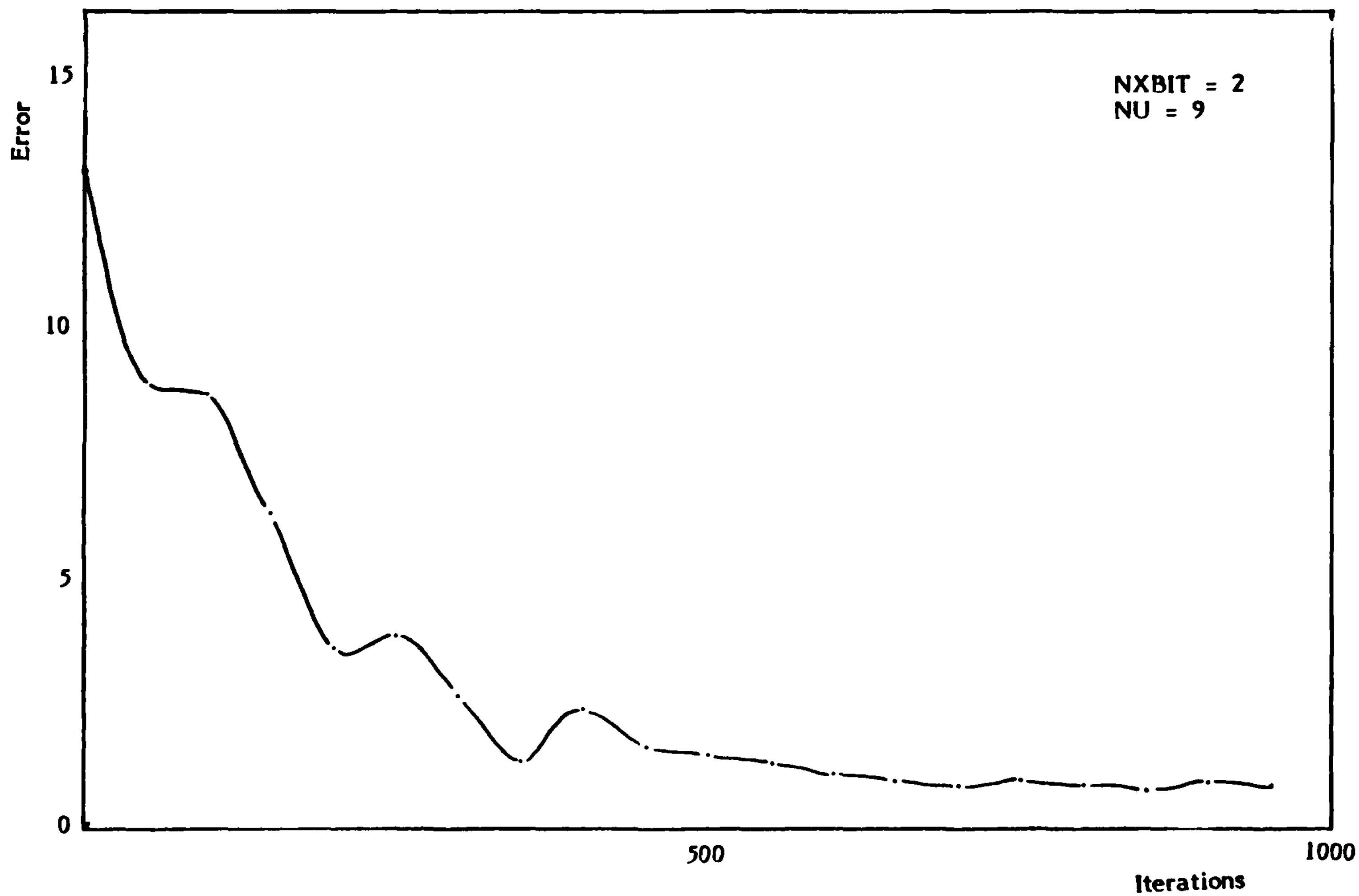


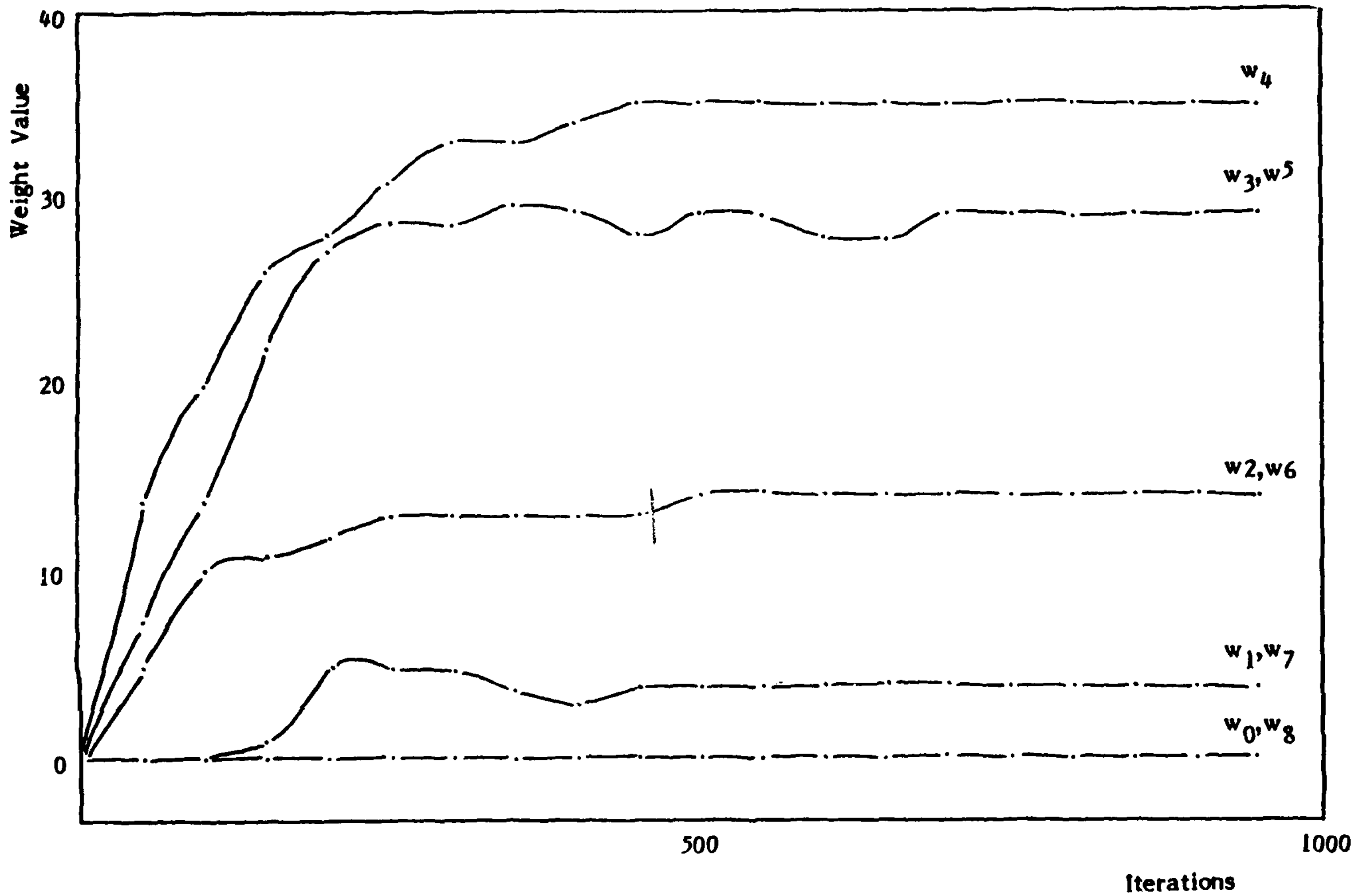
40
30
20
10
0
Weight Value
500
1000
Iterations
w_4
w_3, w_5
w_2, w_6
w_1, w_7
w_0, w_8

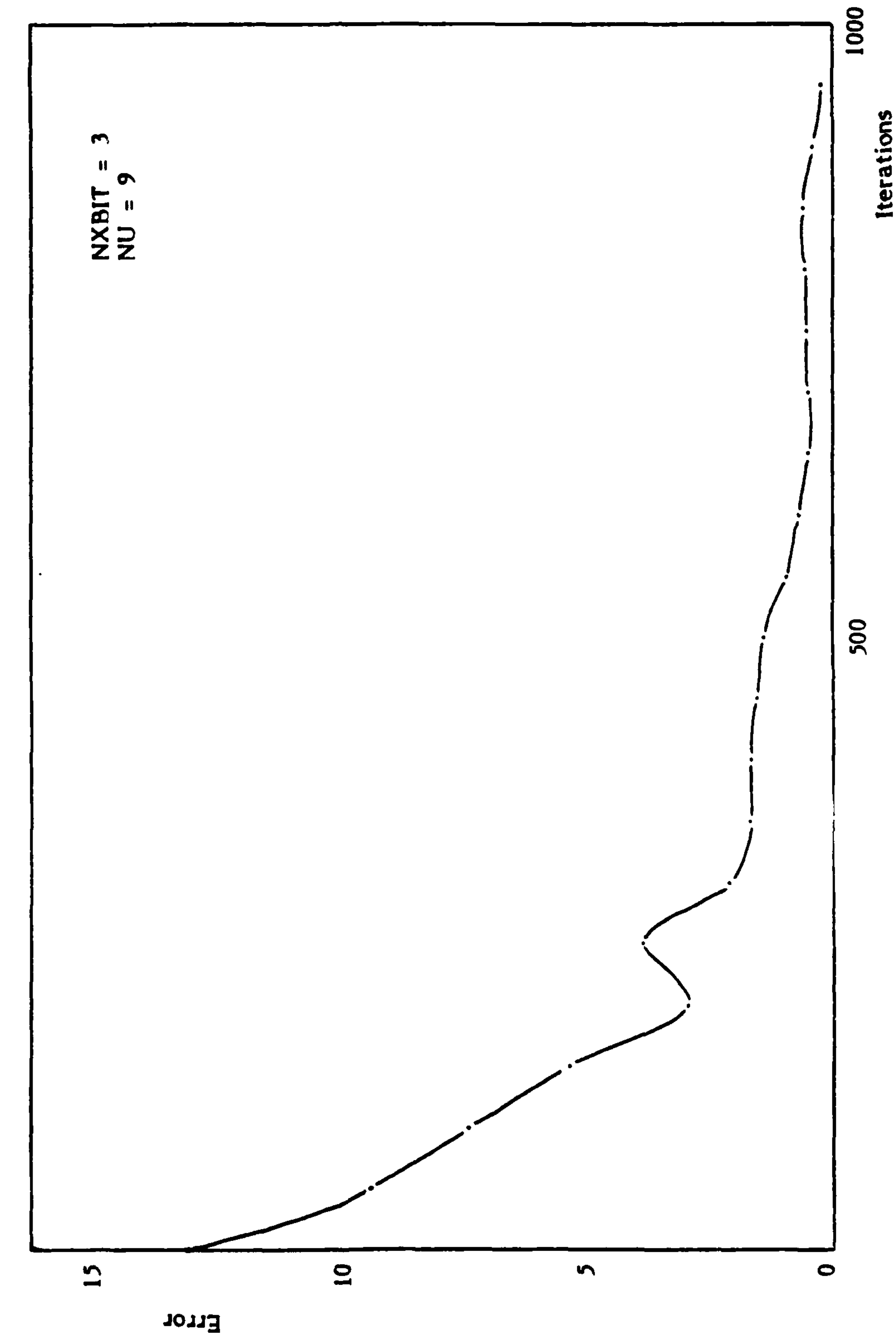

NXBIT = 1
NU = 9
Error
15
10
5
0
500
1000
Iterations



40
30
20
10
0
Weight Value
500
1000
Iterations
w_4
w_3, w_5
w_2, w_6
w_1, w_7
w_0, w_8


NXBIT = 2
NU = 9
Error
15
10
5
0
500
1000
Iterations

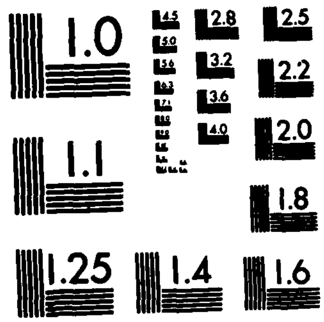

Weight Value
40
30
20
10
0
500
1000
Iterations
w_4
w_3, w_5
w2,w6
w_1, w_7
w_0, w_8


NXBIT = 3
NU = 9
Error
15
10
5
0
500
1000
Iterations

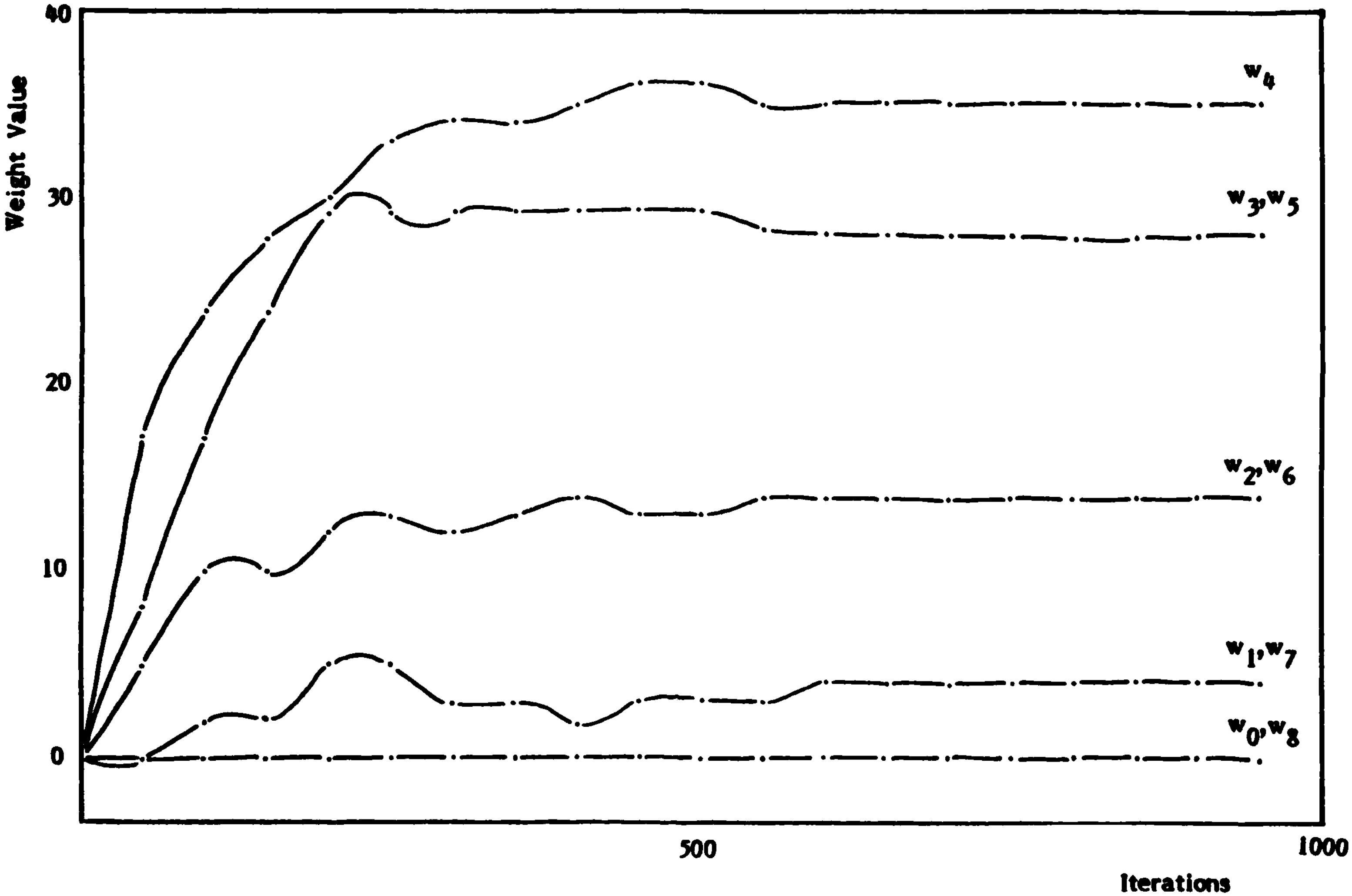
AD-A123 146 INFLUENCES OF HARDWARE IMPLEMENTATION ON A HIGH SPEED DIGITAL ADAPTIVE FI..(U) CALIFORNIA UNIV DAVIS SIGNAL AND IMAGE PROCESSING LAB K D WEINMANN APR 82 SIPL-82-4 2/2

UNCLASSIFIED AFOSR-TR-82-1085 AFOSR-80-0189 F/G 9/1 NL

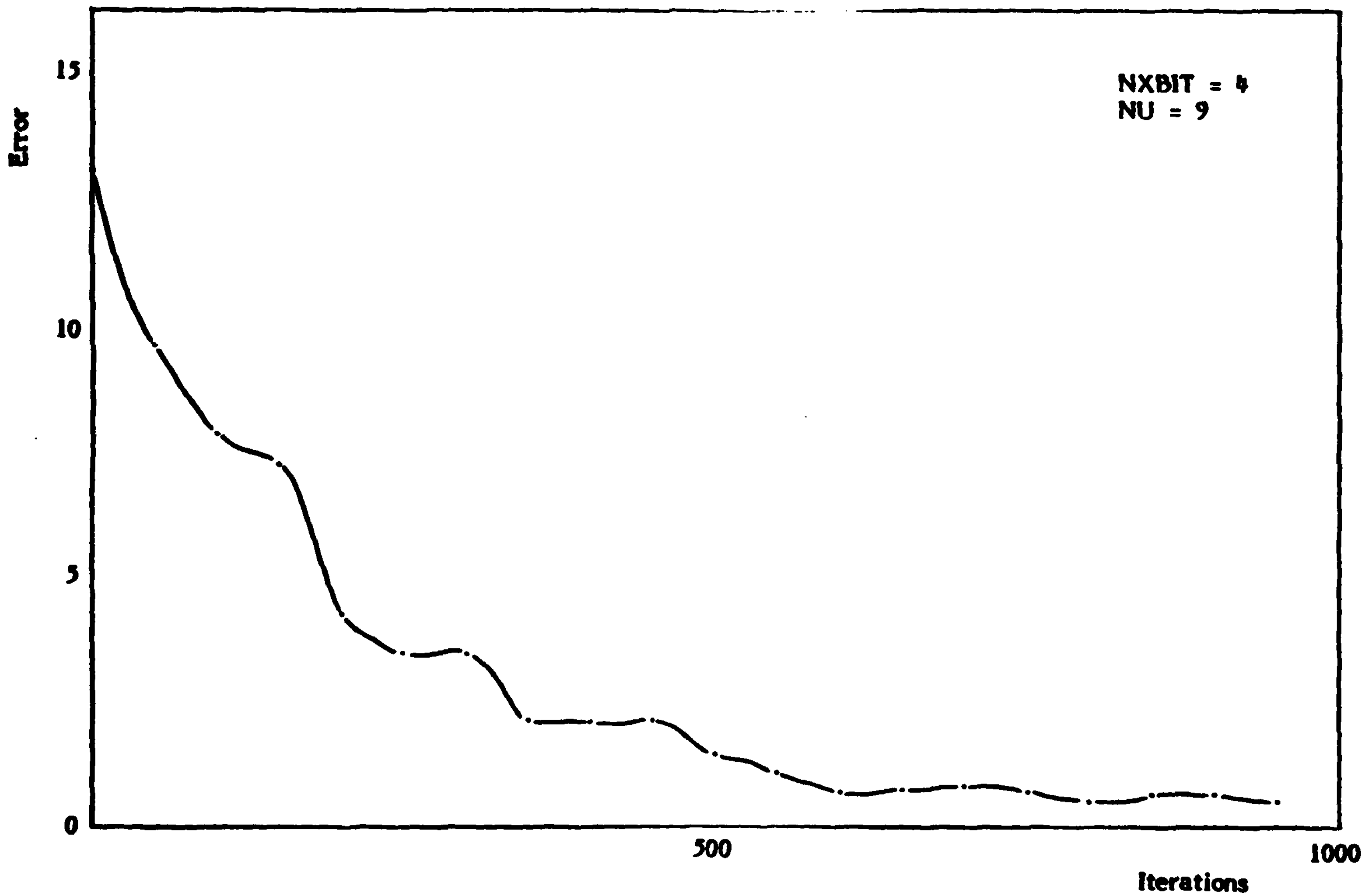
END
DATE
FILMED
2 83
DTIC



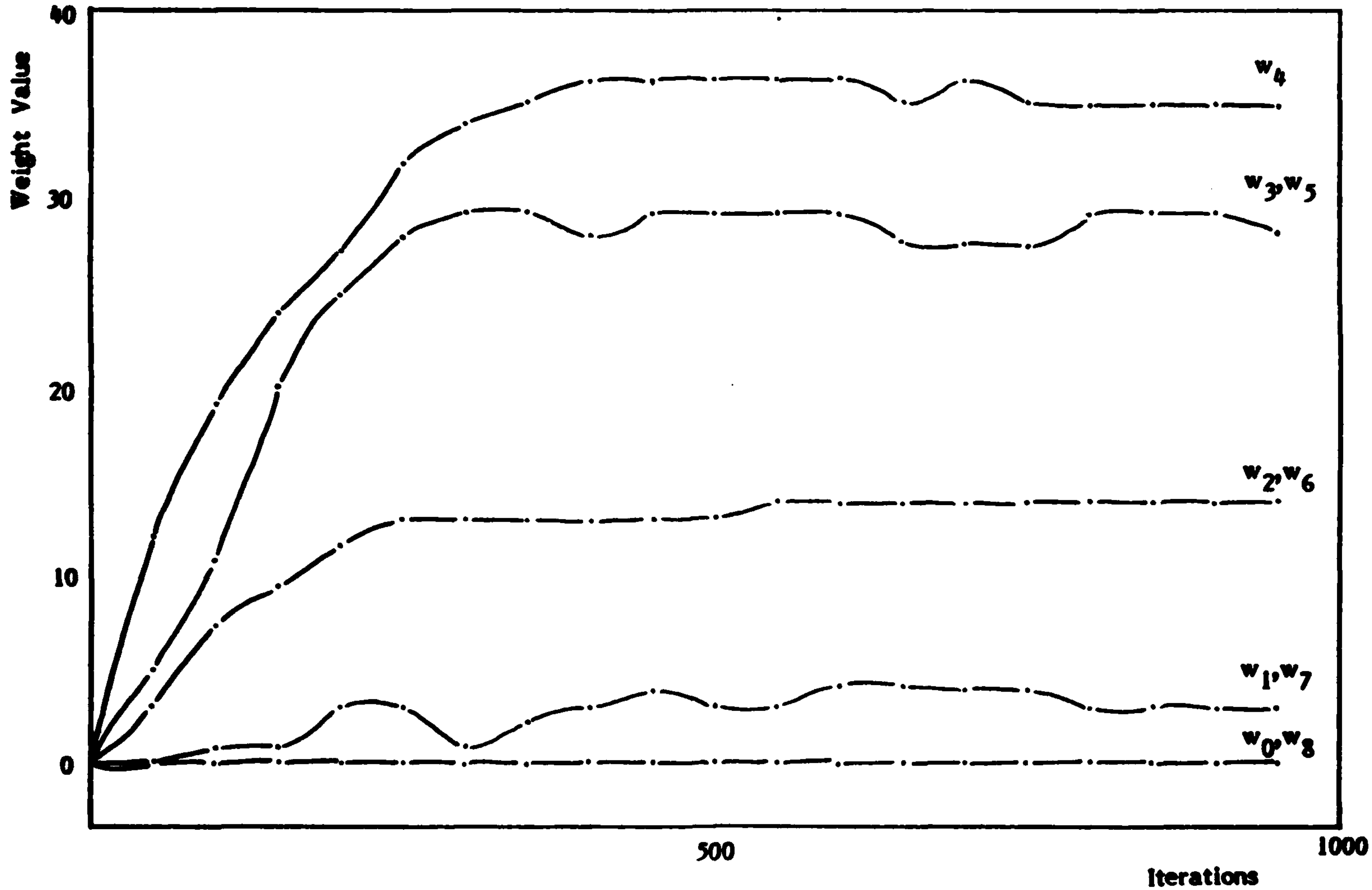
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

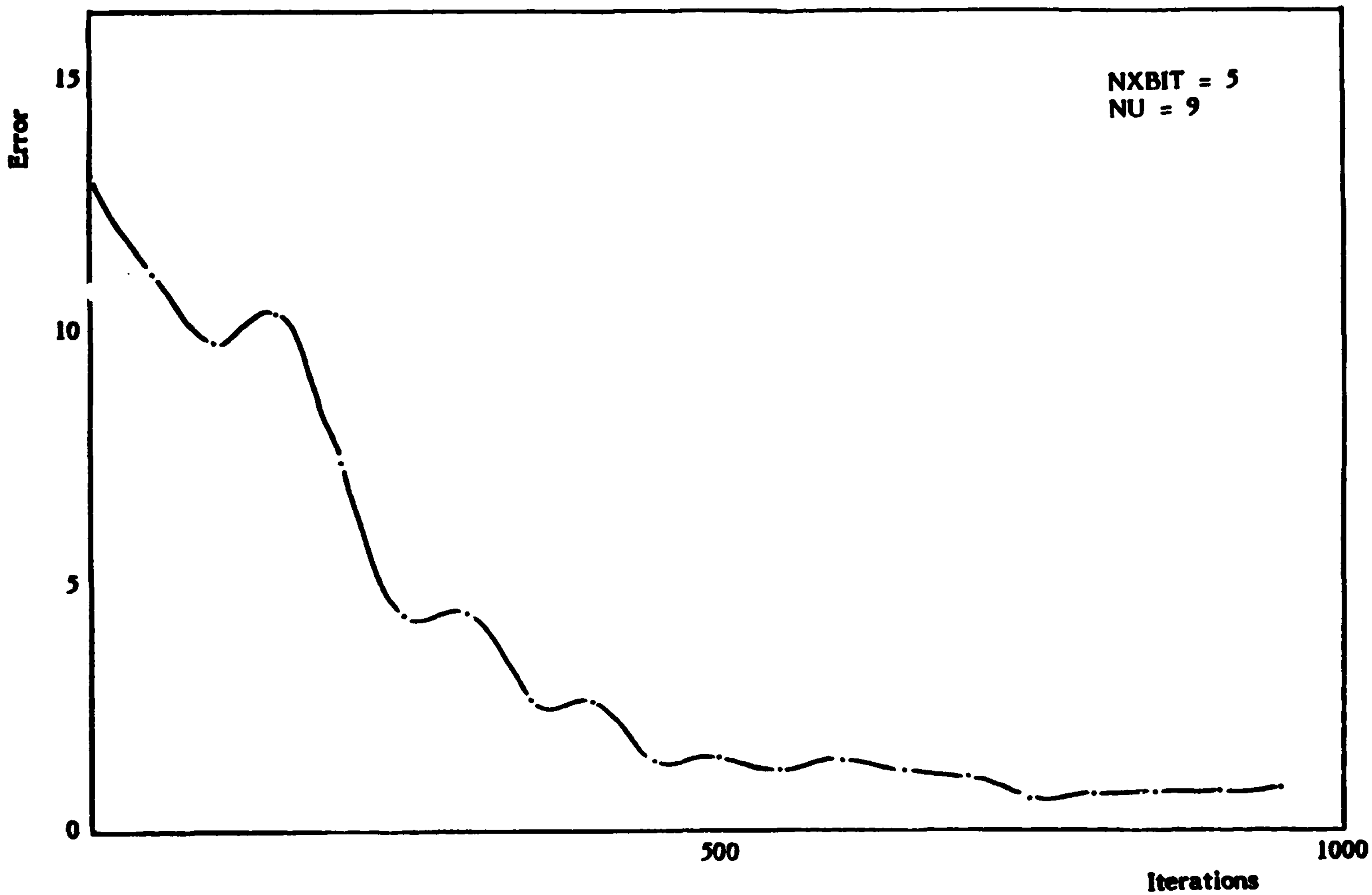


Weight Value
40
30
20
10
0
500
1000
Iterations
w_4
w_3, w_5
w_2, w_6
w_1, w_7
w_0, w_8

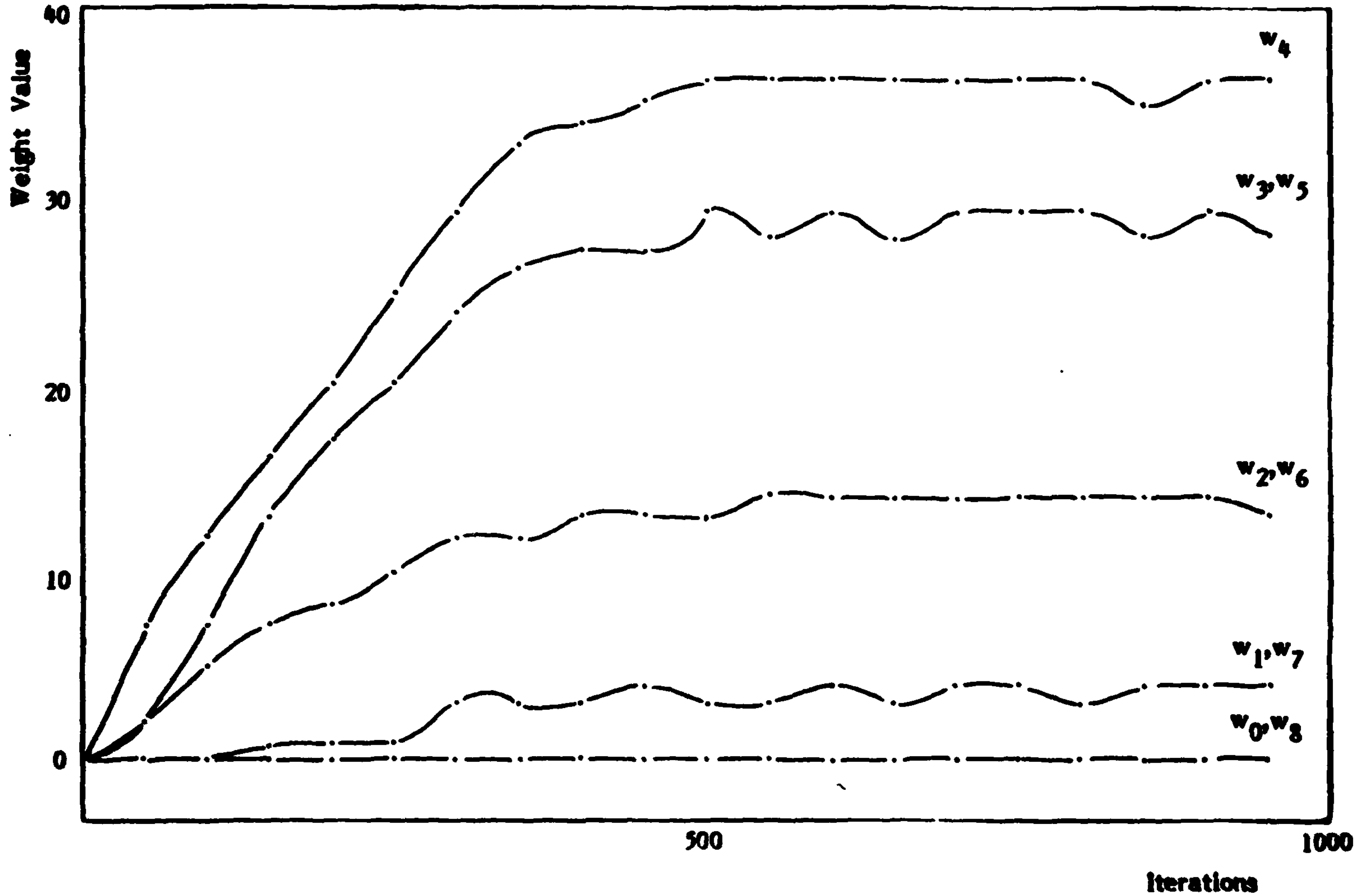


NXBIT = 4
NU = 9
Error
15
10
5
0
500
1000
Iterations

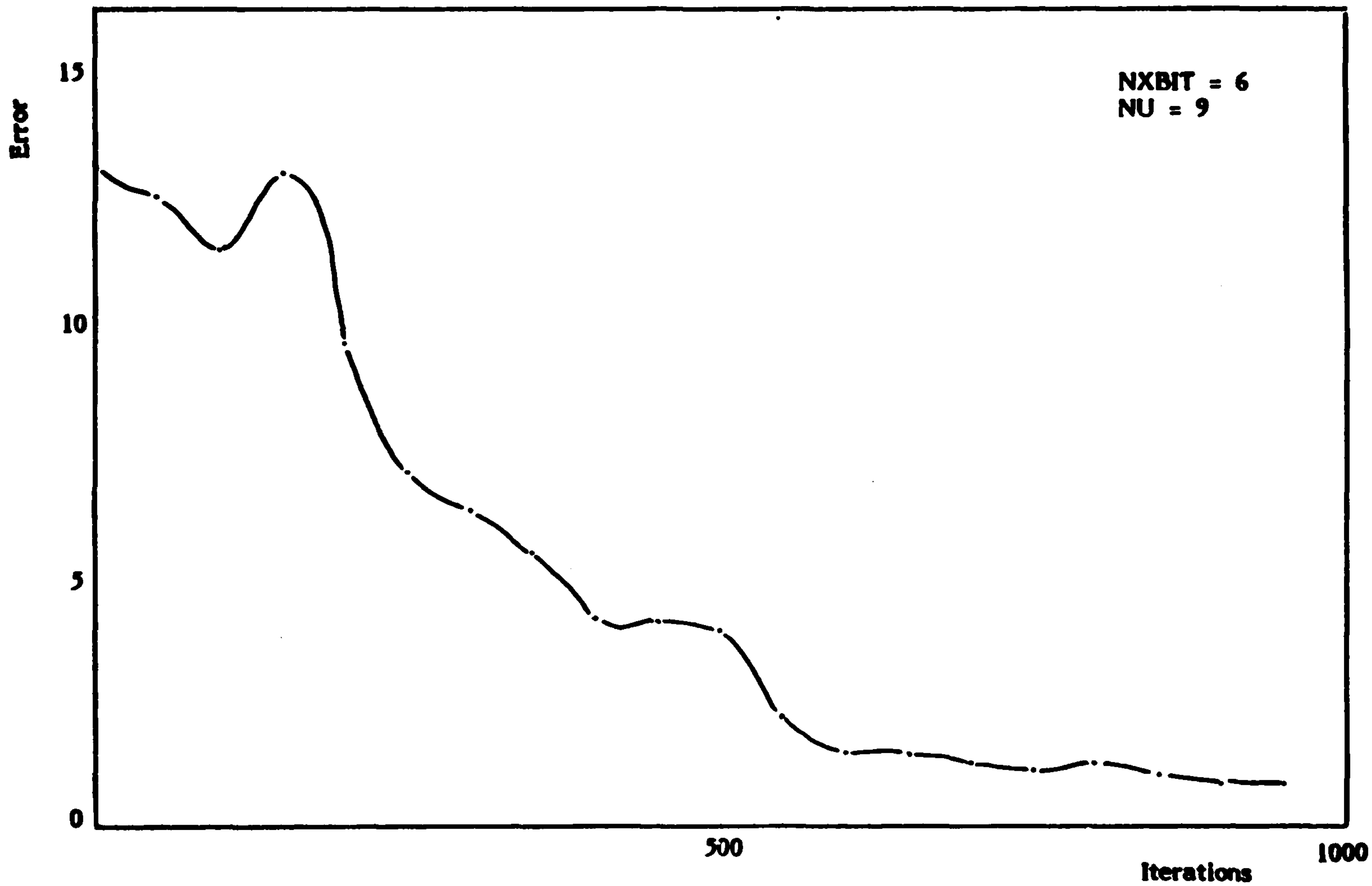

Weight Value
40
30
20
10
0
500
1000
Iterations
w_4
w_3, w_5
w_2, w_6
w_1, w_7
w_0, w_8

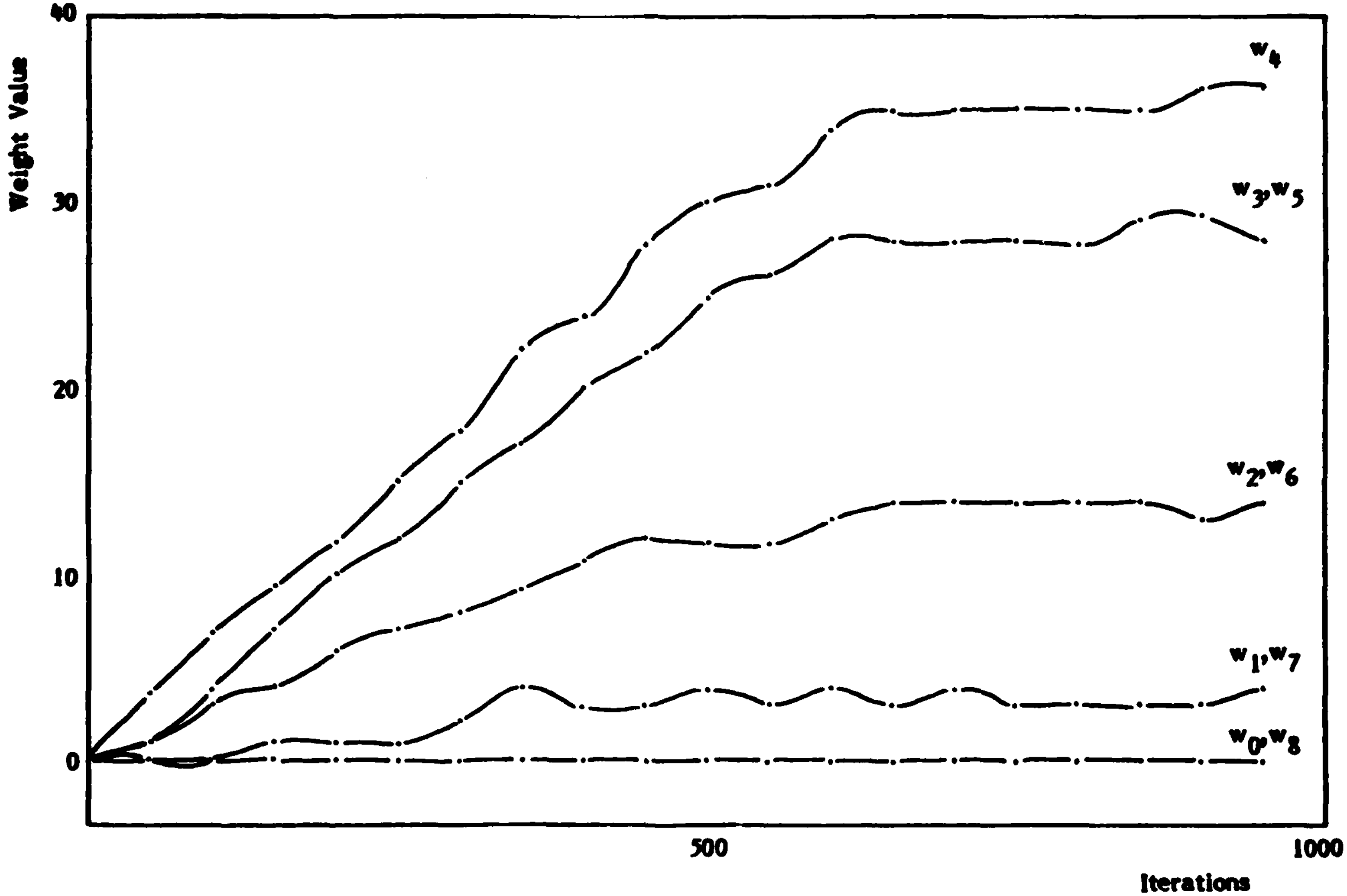


NXBIT = 5
NU = 9
Error
15
10
5
0
500
1000
Iterations



40
30
20
10
0
Weight Value
500
1000
Iterations
w_4
w_3, w_5
w_2, w_6
w_1, w_7
w_0, w_8


NXBIT = 6
NU = 9
Error
15
10
5
0
500
1000
Iterations



Weight Value
0
10
20
30
40
500
1000
Iterations
w4
w3,w5
w2,w6
w1,w7
w0,w8

LME

8